5-2013

Knowledge Extraction in Video Through the Interaction Analysis of Activities Knowledge Extraction in Video Through the Interaction Analysis of Activities

Omar Ulises Florez
Utah State University

KNOWLEDGE EXTRACTION IN VIDEO THROUGH THE INTERACTION ANALYSIS OF ACTIVITIES

by

Omar U. Florez

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Computer Science

Approved:

Dr. Curtis Dyreson Major Professor	Dr. Dan Watson Committee Member
Dr. Xiaojun Qi Committee Member	Dr. Vladimir Kulyukin Committee Member
Dr. Yan Sun Committee Member	Dr. Mark R. McLellan Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2013

ABSTRACT

Knowledge Extraction in Video Through the Interaction Analysis of Activities

by

Omar U. Florez, Doctor of Philosophy

Utah State University, 2013

Major Professor: Dr. Curtis Dyreson
Department: Computer Science

Video is a massive amount of data that contains complex interactions between moving objects. The extraction of knowledge from this type of information creates a demand for video analytics systems that uncover statistical relationships between activities and learn the correspondence between content and labels. However, those are open research problems that have high complexity when multiple actors simultaneously perform activities, videos contain noise, and streaming scenarios are considered. The techniques introduced in this dissertation provide a basis for analyzing video. The primary contributions of this research consist of providing new algorithms for the efficient search of activities in video, scene understanding based on interactions between activities, and the predicting of labels for new scenes.

(154 pages)

PUBLIC ABSTRACT

Knowledge Extraction in Video Through the Interaction Analysis of Activities

Omar U. Florez

A video is a growing stream of unstructured data that significantly increases the amount of information transmitted and stored on the Internet. For example, every minute YouTube users upload 72 GB of information. Some of the best applications for video analysis include the monitoring of activities in defense and security scenarios such as the autonomous planes that collect video and images at reduced risk and the surveillance cameras in public places like traffic lights, airports, and schools.

Some of the challenges in the analysis of video correspond to implement complex operations such as searching of activities, understanding of scenes, and predicting of new content. Those techniques must transform a video into a collection of more informative structures (features, objects, and models) to be analyzed by a computer by considering the relationships of causality and co-occurrence between hidden and observable variables.

The research presented in this dissertation proposes and demonstrates novel algorithms for automatic and semi-automatic analysis of activities in video. The analysis of video is approached with machine learning, databases, and information retrieval algorithms to find repeating patterns and extract co-occurring activities and label them appropriately. The experiments demonstrate that the proposed techniques overcome the limitations associated to each of those operations in a better way than existing techniques.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

CPM Crossdomain Probabilistic Model. A machine learning algorithm that projects user preferences from a source to a target domain.

CF Collaborative Filtering. A recommendation algorithm that predicts user preferences for unrated items by considering other users with similar preferences. An item-based collaboration filtering is also possible, in which case the goal is to recommend to a user a certain item based on other items with similar user distributions.

DTW Dynamic Time Warping. A measure of similarity for two timeseries which may vary in time or speed. It finds an optimal match between two sequences that can be non-linear in the time dimension.

iSAX Indexable Symbolic Aggregate Approximation. A multi-resolution symbolic representation that allows indexing timeseries

HDP Hierarhical Dirichlet Process. A non-parametric Bayesian approach to clustering grouped data. Each group of data is modeled with a Dirichlet Process and all groups share a base distribution drawn from a Dirichlet Process as well.

HRG Hyperspherical Region Graph. A graph data structure that index points in an Euclidean space based on a metric similarity. Hyperspherical regions of fixed capacity are used to contain a number of points that are similar to each others.

HMM Hidden Markov Model. A statistical machine learning algorithm that models a process as a Markov process with hidden states and observed outputs. Each state has a probability distribution over the possible outputs and the transitions between states are parameters in the model.

MAP Mean Average Precision. A measure of precision that averages the individual precision scores of retrieval queries.

LDA Latent Dirichlet Allocation. A probabilistic graphical model that models co-occurrence of words in text documents.

LSH Locality Sensitive Hashing. A hashing algorithm that maps similar vectors in the Euclidean space to the same buckets with high probability.

PCA Principal Component Analysis. A mathematical algorithm that extracts the linearly uncorrelated variables (principal components) of a dataset. The first principal component has the largest possible variance and the next components have the highest variance under the constraint that they are orthogonal. It is commonly used to reduce the dimensionality of a collection of vectors.

SVD Singular Value Decomposition. A factorization algorithm that decomposes a matrix M into three components: U (m×m), Σ (m×n), and V (n×n). The diagonal entries of Σ are the singular values of M. SVD has useful applications in computing the rank, range, approximation, and null space of a matrix.

SVM Support Vector Machine. A supervised learning model used for classification and regression. Given a set of training points, SVM computes a gap between points of different categories as wide as possible. The categories of new points correspond to which side of the gap they fall on.

TSH Timeseries Sensitive Hashing. A hashing algorithm that maps timeseries of similar shape but different length to the same buckets with high probability.

CHAPTER 1

INTRODUCTION

1.1 Introduction

The increasing level of interest for videotaping human activity has produced a massive amount of data. For example, every minute YouTube users upload 48 hours of video [1], which is around 72 GB per minute and 103 TB per day. The ever increasing volume of data creates an urgent demand for video analytics systems in government and industry, in particular for defense and security, health care, and surveillance. For instance, the Army, Navy and Air Force, and the Department of Defense, account for 58.4% of all federal spending on data storage ($1.02B) to mainly store video [2]. Applications of video analytics in security include autonomous planes that collect video and images at reduced risk, surveillance cameras in public places like traffic lights, airports, and schools, and automatic analysis of videos shared on YouTube, Facebook, Twitter, and other online sharing websites. The advances in medical science have created medical diagnosis systems that have massive databases of electrocardiogram, tomography, and monitoring videos to provide automatic diagnosis and treatment while reducing the variance of subjective interpretation and human error [3]. On the industry side, mobile consumption of video is growing dramatically as home bandwidth increases. Algorithms for video analysis deliver relevant videos to viewers based on their preference and video content. Example includes websites that recommend videos such as Netflix and YouTube.

The process of analyzing video into higher-level structures that progressively add more information and value is shown in Figure 1.1. To a computer, an image is a 2D array of pixels with discrete numerical values for color; and a video is a sequence of frames ordered by time. To support searching, scene understanding, and predicting in video, a video (en-

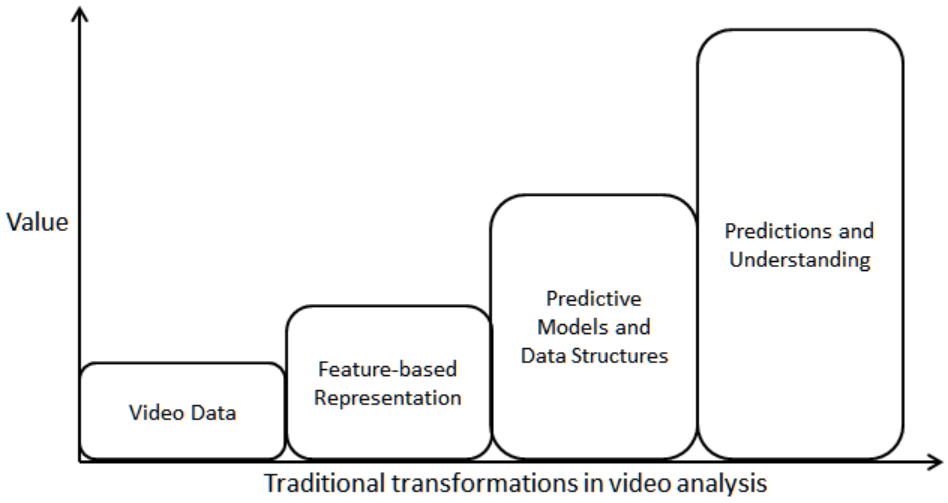


Figure 1.1. Value associated to different transformations for video.

coded as an array of pixels) must be transformed into a collection of higher-level structures (features, objects, and models) describing activities made by moving objects. These abstractions can then be analyzed by a computer by considering the relationships of causality and co-occurrence between hidden and observable variables and modeled with statistical, data mining, and machine learning algorithms. As depicted in Figure 1.1, predicting and scene understanding are the transformations that add the greatest value to the goal of analyzing video. However, those problems are open and have high complexity when multiple actors simultaneously perform activities, videos contain noise, and streaming scenarios are considered.

In this dissertation, we present novel algorithms for automatic and semi-automatic analysis of activities in video. The analysis of video content is approached with machine learning, databases, and information retrieval algorithms to find repeating patterns and extract co-occurring activities and label them appropriately.

1.2 Video Analysis

The analysis of video is composed of different tasks, each with particular goals and algorithms. Figure 1.2 depicts the primary steps made when analyzing video, those that

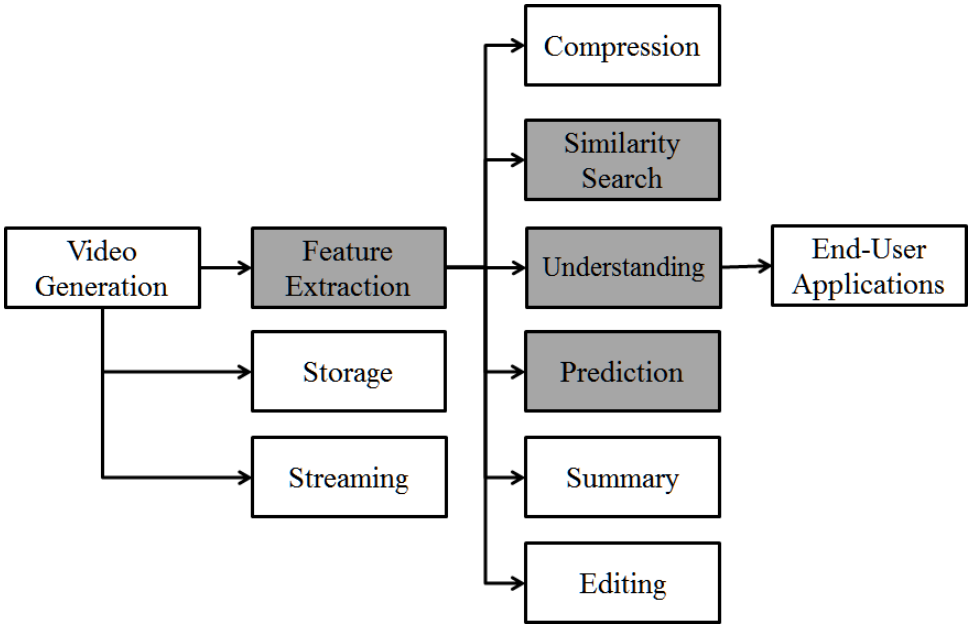


Figure 1.2. Traditional tasks for analyzing video. Blocks that are shaded represent tasks that received contributions from this dissertation.

are shaded represent steps that received contributions from this dissertation.

We define the eleven steps as follows.

1. Video Generation: A video consists of a sequence of frames that records activities at a specific frame rate. Important operations in this step include video encoding and color management.

2. Feature Extraction: This step consists of representing important information from video with significant characteristics of less complexity or dimensionality. Examples of significant features include timeseries to model the temporal behavior of moving objects, gradients to describe important spatio-temporal points associated to high variation of motion, and key frames that model the most important scenes during a video.

3. Storage: This step focuses on the efficient serialization of features in secondary memory. This is important as videos are large and their size is proportional to the duration of the recording.

4. Streaming: Videos are too large to fit in main memory, but they can be streamed in real time so only a small portion is temporally stored in main memory for analysis. Algorithms in this step define time windows in the stream and count subsets of features over time. The distribution of features usually changes over time (a problem known as *concept drift*), so the importance of existing subsets is updated in terms of duration and age.

5. Compression: Repeated patterns, activities, or features in video can be indexed and efficiently stored in memory to reduce the size of a video's content. Continuous features can be described using symbolic approaches to reduce the number of features needed to represent activities in video.

6. Similarity Search: This step accounts for the need to make queries on the video content. Noise usually distorts the feature extraction and therefore affects the representation of activities. Thus, similarity search algorithms quickly retrieve patterns similar to a given query pattern in a short time with a fixed degree of uncertainty. Examples of similarity queries are range and K-nearest neighbors queries.

7. Understanding: The goal of this step is to model how different moving objects interact in a video. Some of these interactions are frequent, while others are only important during the duration of a time window. An interaction can be modeled with random variables that describe relationships of dependency or co-occurrence.

8. Prediction: Video frames or time windows can be labeled by users. For example, some parts of a surveillance recording could be annotated as dangerous scenes. The correct label prediction for new scenes based on prior scenes is the goal of this step.

9. Summary: Videos are large but often have repeated scenes. They can be summarized by a shorter number of important frames. The importance of a frame can be measured in terms of how much information it contains such as interesting events or activities.

10. Editing: The changes on the features that describe activities in a video allow us to generate new content. Examples include actor replacement and automatic scene removal.

11. End-User Applications: This step includes concrete applications that use the algorithms in the above steps to provide interesting services such as pedestrian detection, self-driving vehicles, automatic detection of accidents, and generation of statistics in soccer games.

1.3 Research Scope

The process of searching, understanding, and predicting activities in video involves many open problems in computing, which lack an optimal method to provide fast and precise results. This dissertation presents the following contributions to different stages of those problems. The contributions have been validated through the peer-review process in important computing conferences in the areas of data mining (ACM CIKM'10, '11, '12), multimedia information retrieval (ACM SAC'09, ACM MIR'10, ACM SIGAPP'12), and databases (DEXA'08).

1. Search: We proposed a hashing algorithm, Timeseries Sensitive Hashing (TSH) [4, 5], and formulated the hypothesis that similar timeseries of different lengths map to the same bucket of a hash table without normalization. Experiments support this conjecture and show that this feature is ideal for large databases as it enables sublinear time algorithms for searching and linear time algorithms for clustering of timeseries. Existing research only considered Euclidean vectors, which have the important restriction of needing the same dimensionality (length) for all of the vectors. The natural step to obtain an Euclidean vector from a timeseries is to normalize the lengths for the existing activities, but normalization removes important temporal information and hinders use of the timeseries in streaming scenarios. The major contribution of our research is to successfully overcome this limitation allowing us to model activities with trajectories of variable length. This makes sense because the duration of similar

activities can vary in a video. For the case of Euclidean vectors, we proposed a novel graph data structure called the Hyperspherical Region Graph (HRG), and showed it to be faster than state-of-the-art tree-based algorithms for similarity search. This is because HRG performs search by going through different neighbors (or subtrees) without backtracking to upper nodes as tree-based approaches do [6].

2. Understanding: We developed a method to model activities in a video [7,8]. Our main hypothesis during this investigation is that video scenes could be modeled by the ways in which words group into topics in a text document. Every scene thus corresponds to a multinomial distribution over a number of independent activities and every activity is a cluster of co-occurring features (trajectories or events). This is similar to how words form topics in a text document. By testing this hypothesis, we find that this approach notably reduces the complexity of the problem as scenes could be described with a finite number of topics, but in different proportions. *A posterior* counting of subsets of those topics in a video stream allowed us to extract rules to understand the most frequent interactions between moving objects in a video.

3. Prediction: We introduced a Probabilistic Graphical Model, the Crossdomain Probabilistic Model (CPM) [9, 10], that automatically finds the topics that model the co-occurrence of trajectories in two parts (domains) of a video. We made the hypothesis that those topics form a common structure of hidden random variables that work like a bridge to transfer the user annotations (labels) from an existing source domain to a new one. Experiments showed that we can improve the prediction of labels if we use content from other domains, which supports our hypothesis. The major contribution of this research is to incorporate the latent variables that explain the statistical dependency of features from other domains to increase the average precision in the prediction of labels for new scenes.

The above algorithms collocate with others that form the state of the art in feature extraction, similarity search, scene understanding, and scene prediction. To describe that relationship, we depict alternatives to Timeseries Sensitive Hashing (TSH), the Crossdomain

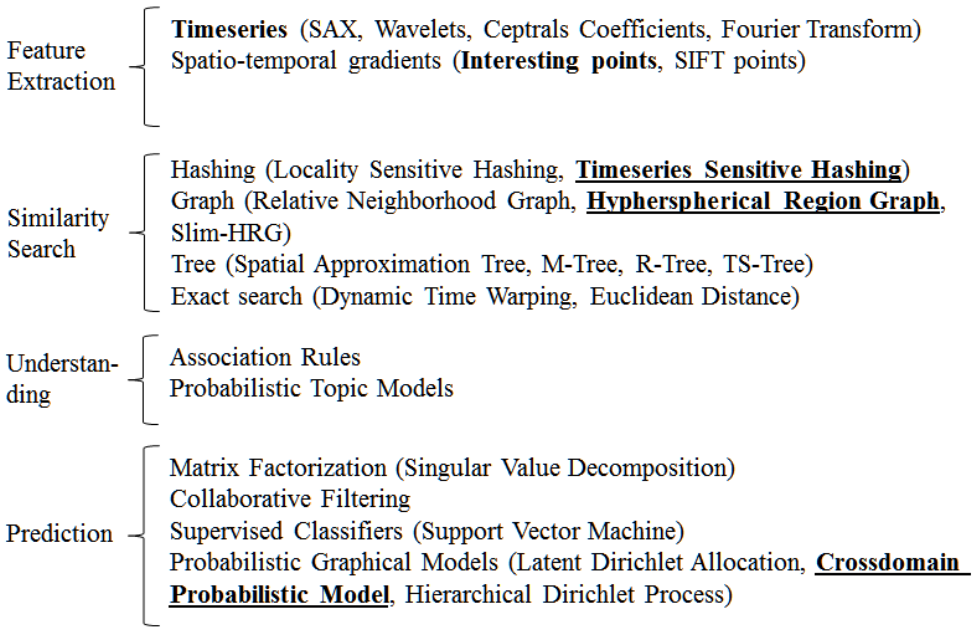


Figure 1.3. A hierarchy of different algorithms to analyze video.

Probabilistic Model (CPM), and the Hyperspherical Region Graph (HRG) for each step in Figure 1.3. In the following chapters, we will describe these algorithms in detail, study related work, and perform extensive experiments to show their *goodness* with respect to similar contenders.

1.4 Example

In this section, we provide an example that shows the outcomes obtained in each of the steps considered in this dissertation. These results are visually depicted in Figure 1.4. The generation of a video consists of recording individual frames over time to capture the interactions of moving objects in scenes. A feature extraction method is needed to find the trajectories of each moving object, which are shown in the first row of images in Figure 1.4. A technique to search trajectories by similarity allows us to group comparable trajectories. This quickly reduces the total number of examples into a vocabulary of discrete elements. The trajectories shown in each figure of the second row of the example correspond to

an element in that dictionary of activities. When multiple objects interact in the same time window, we can measure their frequency of co-occurrence over time. We model this with random variables that groups co-occurring activities into topics, such that a scene is generated by a multinomial distribution over those topics. The frequency of subsets of topics co-occurring in time windows is a value that dynamically changes in the video and it allows us to rank subsets of activities by importance, as shown in the third row of the example. Finally, the prediction of scene labels corresponds to computing the probability of assigning a label for a new scene considering the interactions in previous parts of the video and their correlation with labels.

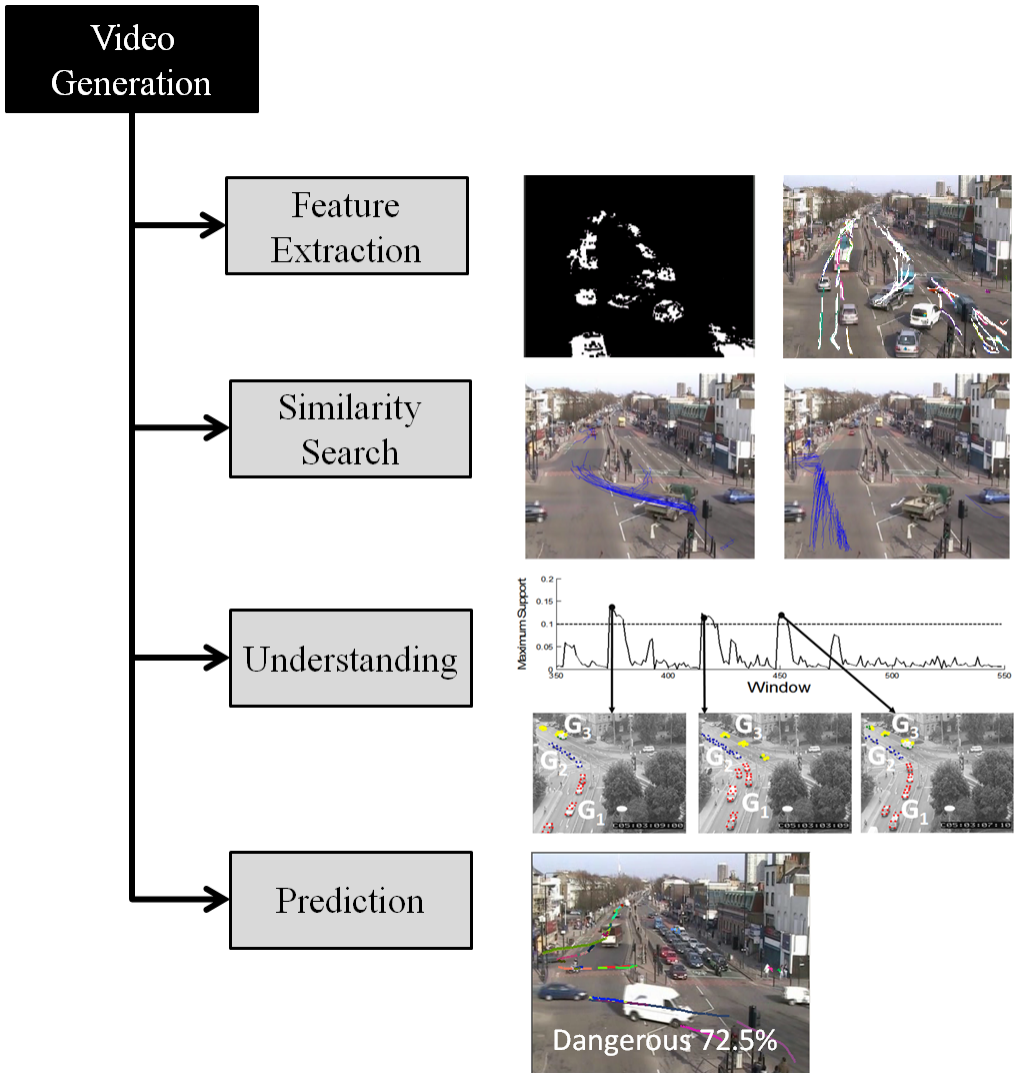


Figure 1.4. An example that shows the outcomes of every transformation for a video.

CHAPTER 2
DISCOVERY OF INTERPRETABLE TIMESERIES IN VIDEO THROUGH DISTRIBUTION OF SPATIO TEMPORAL EVENTS

2.1 Introduction

Motion capture databases are large and widely used in computer animation, robotics, physiotherapy, and sport analysis. Since these databases have traditionally stored motions as time series, many recent efforts have been focus on rapidly indexing [11], comparing [12], and querying [13, 14] the time series associated with moving objects. However, we are still at a point where the original motion information comes from frequently expensive motion capture hardware devices [15], such as the one shown in Figure 2.1. Hence, most research has been constrained to work with public datasets freely available on the web. In response to the lack of affordable hardware, we present in this chapter a method to extract interpretable time series directly from videos. In contrast to the time series extracted using traditional methods, our time series are not based on the tracking of physical points, but describe the motion performed in a video stream as the distribution of spatiotemporal events over frames.

Existing approaches to the expression of human motion as time series have been motivated by the need for measuring the performance of moving objects as one-dimensional signals. Although different types of motion capture devices have been employed for such a goal all have been based on the temporal reading of sensors placed on specific parts of the body. In [17], Knight et al. use accelerometers in a wearable system to analyze and search optimal sporting movements. In [18], Hsu et al. align time series acquired from a motion capture device to produce realistic translations of style in human motion anima-

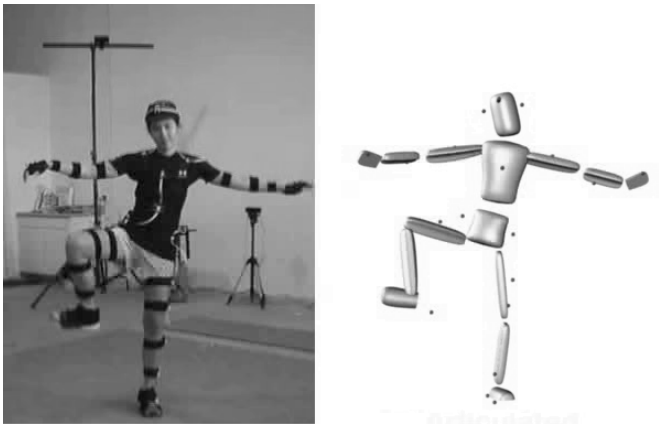


Figure 2.1. Hardware-based motion capture. Left: Motion capture sensors in the human body. Right: The small dots represent sensors placed on specific parts of the person's body. Values of these sensors are then tracked over time to generate time series [16].

tion. However, although time series acquired from sensor readings are generally accurate, the ubiquitous presence of videos makes them attractive as potential sources for motion analysis. As an example, Keogh et al. [19] generate one-dimensional time series from video surveillance data by tracking the right hand of a person over time with the goal of matching the set of time series where the person points with a gun to a target during one second. In contrast to this approach, ours does not require a learning process to identify specific motion shapes through video frames since it only recognizes the abrupt intensity changes made in a local spatiotemporal neighborhood within the input video.

In this chapter, we extract interpretable time series from a widely-available source of motion video. Based on the assumption that natural motion is typically variable, but contains some periodicity over time, we want to generate time series that precisely capture the frequency and amplitude of the movement performed in the video. We experimented with human movements in video and capture the motions by using spatiotemporal gradients as descriptors. Rather than learning models from hand-labeled shapes or events in sequences of frames, we attempt to first describe the motion of each video and then cluster similar motions based on the time series extracted from videos and not from a raw sensor stream. Subsequently, we were able to categorize a set of 114 videos that organizes human motion

into two categories (hand-based and foot-based human motion) with a precision of 86.0% and 75.9%.

The rest of this chapter is organized as follows. Section 2.2 reviews related work. Section 2.3 describes the detection of spatiotemporal gradients within videos. Section 2.4 presents the method to convert the discrete sequences of gradients into continuous time series. In Section 2.5, we show how the time series extracted with this method can be used to categorize human motion and the problems associated with clustering variable length time series. Finally, we conclude in Section 2.6.

2.2 Related Work

Much research has been done to extract information from video data. Some of this research provides methods to make a structural representation of videos. Early work by Kobla et al. [20] extracts a trail of points in a three dimensional space through reduction of dimensionality of the Cosine transform coefficients for each frame of a video. In [21], Fleischman et al. represent hierarchical patterns of temporal information as events from a sequence of frames. These events are then classified to discriminate video content. Additionally, Schuldt et al. [22] also use spatiotemporal gradients to recognize actions from video data. However, this approach requires prior knowledge, i.e., a set of hand-labeled videos, to train the SVM classifier, and widely depends on the spatiotemporal position of each gradient point. Regrettably, such information varies in presence spatial distortion like movement of the camera and zoom in/out operations. More recently, Lai et al. [23] introduce a motion model to represent animal motion from surveillance videos by using the difference of consecutive frames as video descriptors.

Although the above-mentioned projects perform well in their respective application domains, they do not describe the entire motion in the video as a continuous one-dimensional function over time.

2.3 Spatiotemporal Gradients

In this section, we present the way we extract spatiotemporal gradients from a video.

Analogous to image analysis, where spatial gradients (or corners) are used to describe shapes, we use spatiotemporal gradients, not to describe spatial shapes, but motions in videos. Note that, in image analysis only spatial components are considered, but in video streams we also deal with the temporal component. Hence, the neighborhood of a point takes values from the same frame and also from previous and subsequent frames. Our intuition is that a motion m carried out in the video v is the combination of spatiotemporal gradients (or interesting points of Laptev and Linderberg [24]) through the frames. These gradients are generally points with large variation in their spatiotemporal neighborhood. The advantages of using this type of features in v are:

- In contrast to global spatiotemporal descriptors [25], the spatiotemporal gradients use local information between image sequences. Hence, subtler spatial variations between frames are detected.
- Description of the motion based on local gradients can deliver more stable results over changes of rotation, scale, and video sampling than other features such as textures, lines, and blobs.

To detect spatiotemporal gradients, we use the spatiotemporal extension of the Harris Corner Detector [26] made by Laptev and Lindeberg in [24]. Given a point $p(x_0, y_0, t_0)$ placed in the position (x_0, y_0) of the image i contained at the frame t_0, we are interested in detecting any point p whose local variations in all directions (x, y, and t) is strong. The variations of $p(x_0, y_0, t_0)$ in the directions x, y and t are represented by the first order spatial and temporal derivatives (∂_x, ∂_y, and ∂_t) of the image intensity i_p at point p:

$$L_x = \partial_x(g(p, \sigma_s^2, \sigma_t^2) \times i_p)$$

$$L_y = \partial_y(g(p, \sigma_s^2, \sigma_t^2) \times i_p)$$

$$L_t = \partial_t(g(p, \sigma_s^2, \sigma_t^2) \times i_p)$$

where σ_s and σ_t represent the standard deviation with respect to space and time respectively. The values of σ_s and σ_t circumscribe the extension of the spatiotemporal neighborhood around the point p, and g is the spatiotemporal Gaussian kernel centered at p defined as:

$$g(p, \sigma_s^2, \sigma_t^2) = \frac{e^{-\left(\frac{(x-x_O)^2}{2\sigma_s^2} + \frac{(y-y_O)^2}{2\sigma_s^2} + \frac{(t-t_O)^2}{2\sigma_t^2}\right)}}{\sqrt{(2\pi)^3 \sigma_s^4 \sigma_t^2}}$$

We use the *second moment matrix* to compute μ, the intensity structure of the local neighborhood of p. μ is a 3-by-3 symmetric matrix composed of the variations of the points $p \in V$ in directions x, y and t.

$$\mu(p) = g(p, \sigma_s^2, \sigma_t^2) \times \begin{pmatrix} L_x^2 & L_x L_y & L_x L_y \\ L_x L_y & L_y^2 & L_y L_t \\ L_x L_y & L_y L_t & L_t^2 \end{pmatrix}$$

Intuitively, the matrix μ contains all the local changes of image intensities in spatial and temporal directions. Our task consists on finding the local maxima in μ above some threshold value. Note that the resulting second moment matrix is noisy and rectangular because it is based on derivatives. Hence, we averaged it with a smooth circular window as the Gaussian weighting function. A good approximation to finding the local maxima in μ is the Harris corner function which combines the determinant and the trace of μ as follows.

$$H(u) = det(\mu) - (1/27) trace^3(\mu)$$

Then, positive values of H correspond to points with high variation of intensity both in the spatial and temporal dimensions. As we are interested in high changes of local variation, there is no room for considering spatiotemporal interest points with small variations. Hence, we take the top-k points to represent the motion exhibited on each video while suppressing irrelevant variations. The process to identify spatiotemporal gradient from a video is summarized in Algorithm 1.

Algorithm 1 Extract_Gradients (Video v, int σ_t^2, int σ_t^2)

1: **for** $t = 1$ to number_frames(v) **do**
2: $image \leftarrow v(t)$;
3: **for** $x = 1$ to width($image$) **do**
4: **for** $y = 1$ to height($image$) **do**
5: $p \leftarrow image(x, y, t)$;
6: $L_x(x, y, t) \leftarrow \partial_x(g(p, \sigma_s^2, \sigma_t^2) * p)$;
7: $L_y(x, y, t) \leftarrow \partial_y(g(p, \sigma_s^2, \sigma_t^2) * p)$;
8: $L_t(x, y, t) \leftarrow \partial_t(g(p, \sigma_s^2, \sigma_t^2) * p)$;
9: $\mu(\text{x,y,t}) \leftarrow g(p, \sigma_s^2, \sigma_t^2) \times \begin{pmatrix} L_x^2 & L_xL_y & L_xL_y \\ L_xL_y & L_y^2 & L_yL_t \\ L_xL_y & L_yL_t & L_t^2 \end{pmatrix}$;
10: **end for**
11: **end for**
12: **end for**
13: $H \leftarrow det(\mu) - (1/27)trace^3(\mu)$;
14: $[g\_values, x, y, t, L_t^2] \leftarrow (H > 0)$;
15: **return** $[g\_values, x, y, t, L_y^2]$;

2.4 Motion as Timeseries

In this section, we show that spatiotemporal gradients can be turned into time series to mine motions. A time series is a sequence of observations made sequentially in time and typically spaced at uniform time intervals. We saw before that given a video v, gradients can locally represent high intensity variation in spatiotemporal neighborhoods. However, a time series representation of v has the advantage in describing the entire motion in a more comprehensive (and one-dimensional) way. Here, we argue that given a video v and a temporal variable e (called an *event* in this chapter), a suitable interval of the values of e defines a shape that can well characterize the type of motion performed in v. Intuitively, this interval of values is designated as the time series TS that characterizes v. We shall first define the way we define and quantify an event e at time t. Then, we provide an example to show how the different values of e over time define a time series.

Definition 1 (event)**.** *Given a video v, the event $e = (t, G, \sigma_t^2)$ is defined by the existence of a set of spatiotemporal gradients $g_i \in G$ within the interval delimited by $[t - \sigma_t, t + \sigma_t)$, where*

1. *t is the number of the frame in the video v where e is placed,*
2. *σ_t^2 is the temporal variance used before to extract gradients,*
3. *The duration of the event is $2 \times \sigma_t$, and*
4. *The value of the event e is defined as $\overline{e} = |G| \times \sum_{i=1}^{|G|} |g_i|$.*

Definition 2 (time series)**.** *Given a set of events $e_t \in E$ whose values depend on time t, the time series TS for the video v is defined as*

$$TS_v = \overline{e}_t \text{ such that } t \in [1, |v|]$$

where $|v|$ is the number of frames present in v.

As an example, Figure 2.2 shows the temporal distribution of events from a person that makes a running motion in a video. Each number in the time-axis represents the current frame in the video stream and the temporal variance is $\sigma_t^2 = 2$ for each frame. Note that, although the total motion performed in the video can be represented as one general time series (TS_4), some similar patterns are repeated over time (TS_1, TS_2, and TS_3). In this chapter, we call these patterns *unit motions*, the minimum pattern in the general time series (TS_4) that captures the nature of the movement present in a sequence of frames. Although several types of *unit motions* can be found if a person performs different types of movements, we assume that we only deal with videos of one type of motion.

From the unit motions shown in Figure 2.2, we can see that a time series relies on three important attributes.

1. Amplitude. The value of this attribute depends on the value of each event, i.e., the product of the number of gradient points used to represent the global time series and the magnitude of change of each gradient associated with an event, as shown in Definition 1. Note that, by taking the most intensive gradients, we assure to consider the gradients associated with regular motions.

2. Frequency. This value represents the number of occurrences of a repeating event per unit motion. Since different temporal variance values lead to define different events,

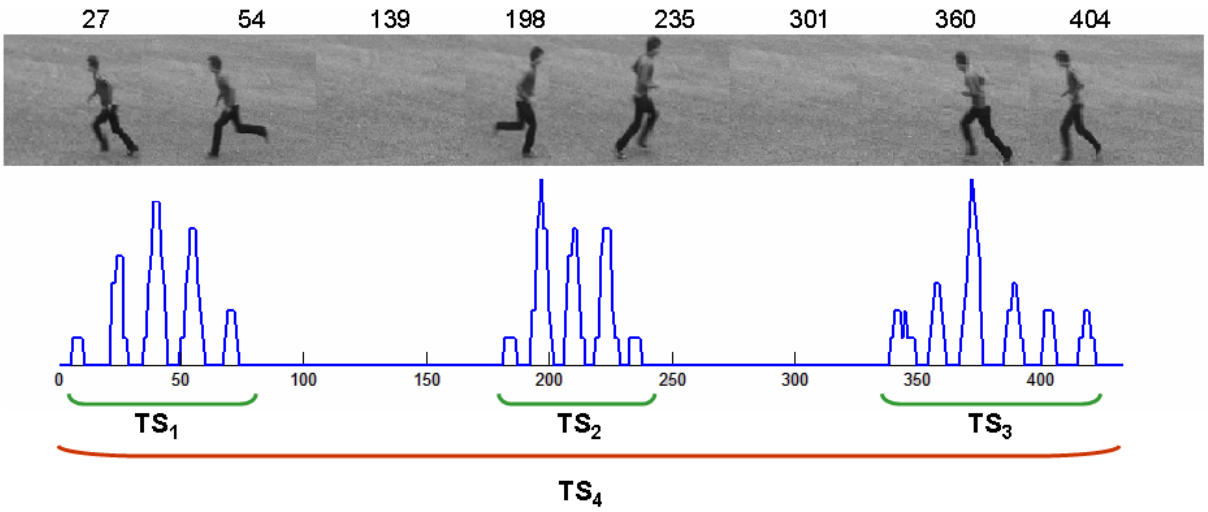


Figure 2.2. Temporal distribution of events from a person that makes a running movement in a video. Note that different time series can be obtained from the video. Although TS_4 represents the total motion in the video, we are interested in detecting *unit motions* (TS_1, TS_2, and TS_3) that comprise comprehensive types of motions during shorter time periods.

we set this value as constant for all the videos. In other words, a different amount of gradient points may fall into one event if we set a different duration for the same set of events.

3. Length. The duration of a time series depends on the duration of the movement performed in v. Hence, we obtain different lengths from two videos even if they contain the same type of movement.

Both the amplitude and frequency comprise the information that differs one time series from another. However, motions are often of different length which makes comparison difficult. We will discuss our approach to solve this problem in the next section.

2.4.1 Acquiring Time Series for Human Motion Interpretation

We are now ready to algorithmically discuss the extraction of human motion time series from video stream data. Given a video v, we are interested in retrieving the k most important spatiotemporal gradients in terms of sharp changes in a local neighborhood whose

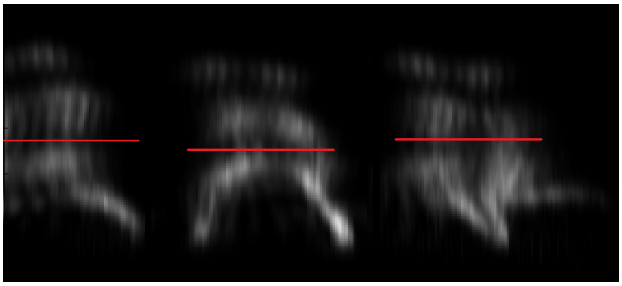


Figure 2.3. Spatial changes of the moving object (in white) with respect to time are used to evaluate the position of the half line that will divide the body of a person into lower and upper regions. Note that the position of the half line varies since the position of the person also varies at each frame. Pixels with variation of intensity close to zero are shown in black.

extension is defined by the squared root of the spatial and temporal variance σ_s^2 and σ_t^2, respectively.

The motion m in v is mainly characterized by the motion exhibited in the upper and lower part of the human body. If we assume that the motion is made perpendicular and horizontal to the camera, a half line that divides the human body into two parts (upper and lower) can be used to describe the global human motion as two time series. Each time series represents the motion performed in the upper or lower part of the body.

For such a goal, we employ the intensity of the spatial variation of each frame with respect to time. This information is contained in the three dimensional matrix L_t^2 and describes the spatial variations of the moving object by considering contiguous frames. If a spatial region has slightly changed over these frames, the value of L_t^2 is close to zero, as is shown in Figure 2.3. Note that, since the $Extract\_Gradients()$ method retrieves L_t^2, we use the information contained in it to evaluate the half lines of the person's body (i.e., moving object) by computing the mean with respect to y of the values of L_t^2 that are not zero in each frame. This line is only defined in the frames that contain spatiotemporal gradients; otherwise its value is zero. The spatiotemporal gradients that are below its corresponding half line are considered gradients that represent the lower part motion. Otherwise, they belong to the type of gradients that represents the upper part motion. Figure 2.4 shows the two types of gradients (upper and lower placed) found in a video.

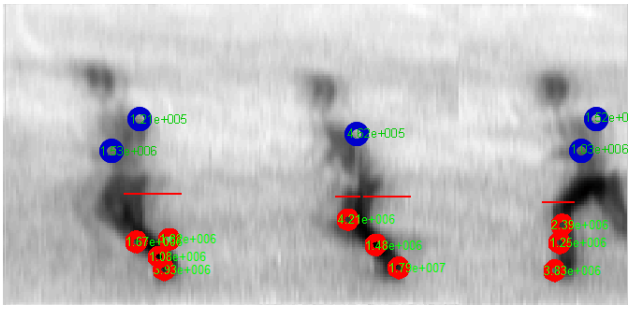


Figure 2.4. By using the half line, we can detect two types of spatiotemporal gradients: lower (in lighter circles) and upper (in darker circles). The length of horizontal line represents the duration of an event. Note how spatiotemporal gradients are defined within the interval represented by the length of this line.

Once we obtain a discrete set of spatiotemporal gradients categorized by their spatial location in the human body, we want to describe the human motion as a continuous function. Instead of counting the number of events per frame and obtaining a coarse sampling of the time series, we follow our definition of events to express the time series as a distribution of events over time. In such a case, given a frame at the time t, we consider the set of gradients $g_i \in G$ that fall in the short-time window of time length equal to $[t - \sigma_t, t + \sigma_t]$ as an event whose value is defined as $|G| \times \sum_{i=1}^{|G|} |g_i|$. By slicing this window from the first to the last frame, we average the number of gradients available in contiguous events and smooth some noise produced by considering an arbitrary temporal variance in the algorithm. The result is a composed time series that convolves the number of gradients detected at each event with the total value of the event. Intuitively, the time series can also be considered as the distribution of event values over time.

2.5 Clustering Timeseries

In this section, we want to organize human motion time series according to similarity. This process is known as clustering in data mining and offers some challenges in the context of human motion.

First, we need to iteratively compare time series of varying length. The length of each time series depends on the duration of the motion attained in each video. Furthermore,

Algorithm 2 Extract_Timeseries (Video v, int σ_s^2, int σ_t^2, int k)

1: $[gradients, x, y, t, L_t^2] \leftarrow$ Extract_Gradients(v, σ_s^2, σ_t^2);
2: $[g\_values,\ x,\ y,\ t,\ L_t^2] \leftarrow max_{1 \leq i \leq k}\{\text{sort}([g\_values, x, y, t, L_t^2])\}$;
3: $half\_lines \leftarrow mean(L_t^2, 2)$; // evaluate the mean of L_t with respect to the second dimension y
4: **for** $t = 1$ to number_frames(v) **do**
5: **for** g in G **do**
6: **if** $t - \sigma_t \leq t_g \leq t + \sigma_t$ **then**
7: **if** $y_g < half\_lines(t)$ **then**
8: $lower\_g(t) \leftarrow lower\_g(t) \cup g$;
9: **else if** $y_g \geq half\_lines(t)$ **then**
10: $upper\_g(t) \leftarrow upper\_g(t) \cup g$;
11: **end if**
12: **end if**
13: **end for**
14: $lower\_timeseries(t) \leftarrow |lower\_g(t)| * \sum_{i=1}^{|lower\_g(t)|} lower\_g_i$;
15: $upper\_timeseries(t) \leftarrow |upper\_g(t)| * \sum_{i=1}^{|upper\_g(t)|} upper\_g_i$;
16: **end for**
17: **return** $[lower\_timeseries, upper\_timeseries]$;

the same person can perform faster or slower movements at different times. Hence, we commonly obtain time series with variable length and amplitude even if we choose two time series of the same type of motion. This fact has a direct effect in clustering tasks because we iteratively measure the dissimilarity between pairs of time series to set cluster memberships and evaluate centroids. In fact, traditional distance functions, like the ubiquitous Euclidean distance, are not suitable in this context since they are based on the pair-wise comparison of two patterns. In other words, Euclidean distance is unable to tolerate small distortions and misalignments in a sequence or compare two sequences of different length.

In response to such issues of Euclidean distance, Dynamic Time Warping (DTW) has been chosen as the preferred dissimilarity function when comparing two time series. However, DTW is still improved in terms of accuracy by first scaling in length and amplitude of the associated time series. This effect has recently been noted by other authors. As an example, Fu et al. in [13] claimed that "it has been shown that in many domains it is also necessary to match sequences with the allowance of a global scaling factor." To compare

two time series (TS_1 and TS_2) of different length ($|TS_1| < |TS_2|$), we avoid pair-wise comparison by scaling the longest time series TS_2 to the smallest one TS_1 by removing the less important coefficients of the Fourier representation of TS_2. The resulting time series $TS_2\prime$ has the same length of TS_1 while still preserves the information contained in the original time series TS_2. This approach leads us to reduce the high error rates obtained by using the Euclidean distance to compare two time series of different length.

Second, although we could compare the global time series extracted from the video (e.g., TS_4 in Figure 2.2), we prefer to compare unit motions for this task. This is because the time series associated with the entire video records both the distribution of gradients over frames and also the silence periods with no motion in the video. Considering unit motions as words in a voice signal and the remainder of the signal as noise or silence gaps serves in this context as an analogy. Since the frames associated with intervals without motion often varies in length, by comparing *words* only, we reduce the error produced by adding variable *silence gaps* in the time series. Thus, if a video only contains one type of movement, each unit motion is an independent portion of total time series that characterizes the video.

To extract unit motions from the video, we use the *Zero-Crossing rate* method heavily used in speech recognition to detect the interval of the signal that contains isolated words. This method segments the time series by returning the start and end points of each unit motion and consists of measuring the rate at which the signal changes from positive to negative with respect to a horizontal level. Since many unit motions may be retrieved, we always choose the first one for later activities. If the time series does not exhibit any unit motion, we consider the frames to contain no motion.

2.5.1 Experimental Results

We tested our proposed method to extract time series from the videos of the Action Database [22] that consists of 114 gray-scale videos of 6 different types of motions (boxing, hand clapping, hand waving, jogging, running, and walking) performed by 19 people. All videos have the same frame size (120 x 160 pixels) and frame rate (25 fps). The number of

frames in the videos varies from 300 to 750 frames and includes 40 minutes of video in total. In our experiments, we assume that the movement is made in an orthogonal and horizontal position with respect to the camera.

For the clustering of the human motion videos, two unit motions (i.e., time series), one from the lower and one from the upper part of the body, are extracted from each video. We use the k-Means algorithm to cluster the pairs of time series with k=2, anticipating that one resulting cluster contains hand-based movements (boxing, hand clapping, and hand waving), and the other foot-based movements (jogging, running, walking). We measure the quality of our approach by measuring the precision of each cluster in each type of motion j, which is defined as follows.

$$precision_k = \frac{N(c_k)}{N(c_j)} \times 100\%$$

where $N(c_k)$ represents the number of videos correctly categorized in cluster k and $N(c_j)$ corresponds to the total number of videos that contains the type of motion j (19 videos per each type). The results of this measure per each type of motion are summarized in Table 2.1.

In the case of the hand-based movement cluster, note that both boxing and hand clapping motions exhibit more accurate results than the hand waiving motion. This is because, in the hand waving motion the occasional presence of hand movement below the half line of the body adds some noise to the pair of time series extracted.

On the other hand, note that the foot-based movement cluster exhibits 11.7% less accurate results on average than the hand-based movement. This is because these types of movements involve motions in both the upper and lower part of the body. Hence, some foot-based movements are recognized as hand-based movements when the upper part movement is dominating.

These results seem to indicate that although our approach of using the half line of the human body to discriminate types of human motion works reasonably well, there is still room for improvement by considering a different method to recognize body regions.

Table 2.1. Precision of the Classification of Human Motion Timeseries from 114 Videos into Two Groups (Hand-based Movement and Foot-based Movement).

Motion type	**Hand-based movement**	**Foot-based movement**
Boxing	89.5%	10.5%
Hand clapping	89.5%	10.5%
Hand waving	78.9%	21.1%
Jogging	28.8%	71.1%
Running	21.1%	78.9%
Walking	22.3%	77.7%

2.6 Conclusion

In this chapter, we propose a new method to model the motions performed in videos as time series. We define the associated time series as a function based on the values of events at different frames. Each event is defined as a set of spatiotemporal gradients that fall in the interval whose extension depends on a given temporal variance. Since gradients represent the set of points with the largest variation, our approach seems robust in the presence of noise and irrelevant motions because it works by only considering the most significant movements in the video.

In conclusion we make two observations. First, by convolving the number and total intensity of the gradient points present during an event, we obtain more discriminative time series than by using these values independently. Second, the intrinsic relationship between spatiotemporal events and frame pixels suggest that more accurate results can be achieved by an attenuation of irrelevant components like moving shadows and backgrounds. Moreover, a more accurate method than using the half line of the human body is desirable to accurately recognize the upper and lower time series from human motions.

As future research, we plan to study the application of the method proposed in this chapter to analyze real life human motion such as sports and gait movement with the goal of optimizing human motion categorization through time series analysis.

CHAPTER 3
FAST SIMILARITY SEARCH OF REALISTIC TIMESERIES EXTRACTED FROM VIDEO

3.1 Introduction

Timeseries are an important representation of the behavior of processes over uniform time intervals. They are used in many, diverse fields such as computer animation, robotics, gene expression, electrocardiograms, stock market quotes, and multimedia data. In this chapter, we focus on an important and interesting special case: the representation of motion recorded from live actors and described as a timeseries. The indexing, querying, and classification of motion-related timeseries is an open problem. The motion of an actor can be visually represented with timeseries. As an example, Figure 3.1 illustrates the use of timeseries to represent human motion. The timeseries are generated by sensors placed on the body of an actor. Each sensor measures an aspect of the motion, for instance angular velocity or spatial position, as the data is collected over time and thus forms a timeseries.

Motion-related timeseries have features that are not commonly present in traditional types of vector data, which create additional indexing challenges as described in more detail below.

High and variable dimensionality: Figure 3.1 shows five timeseries generated by each actor (hands, feet, and head) during individual trials to record human motion. For different trials and actors, the duration of the motion *varies*. It must be large enough to represent different kinds of motions (e.g., walking, running, jumping, etc.) and each motion could have a different duration. If we consider these timeseries as vectors whose dimensionalities are a function of the motions' duration, we will obtain vectors of *high* and *variable* dimensionality. We call these patterns *realistic timeseries* to differentiate them

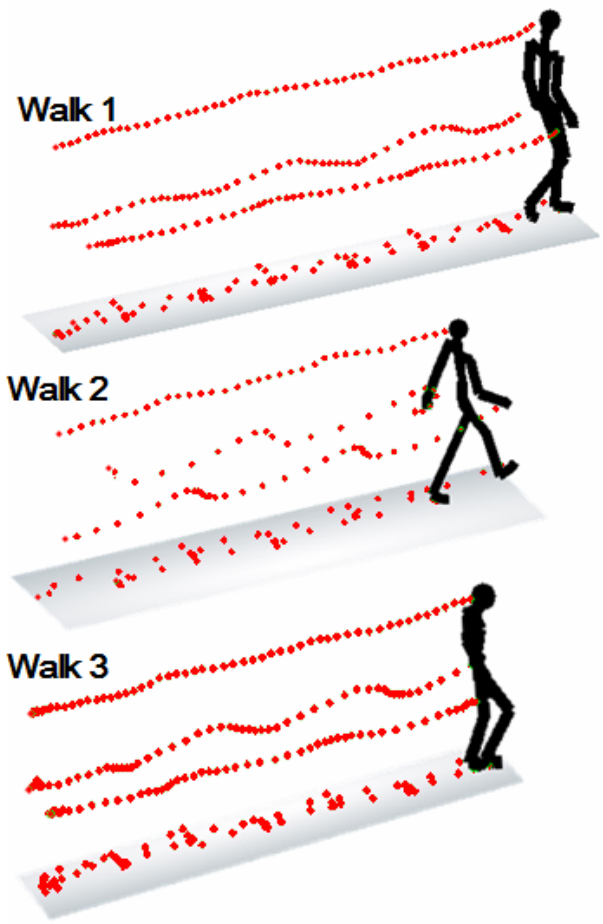


Figure 3.1. Timeseries generated from human motion as different actors move. Note that each timeseries is formed by reading the values of one sensor over time.

from other types of data. Figure 3.2 shows three realistic timeseries that are part of a dataset of N patterns. Both features (high and variable dimensionality) are common in *human motion databases* and are the main obstacles to indexing timeseries with traditional data structures. Realistic timeseries are studied in this chapter and are also found in other contexts such as speech recognition [27], stock market quotes [28], figure shapes [29], and query by humming [30].

No Euclidean distance: In a timeseries, the n^{th} value represents a measurement of a process for the n^{th} time interval, which is a direct result of its preceding values. The Euclidean distance between two timeseries can be computed by pairing up values from each

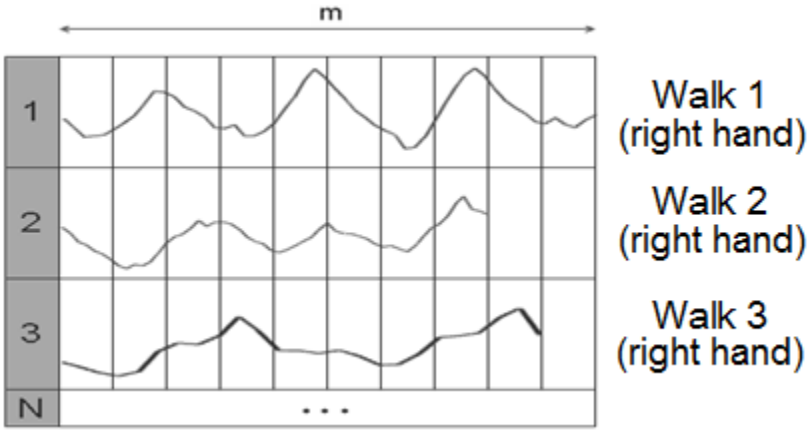


Figure 3.2. Timeseries of different lengths have variable dimensionality. In this case, the longest timeseries has m dimensions.

timeseries. But timeseries of variable dimensionality cannot be aligned pair-wise, so computing the Euclidean distance is problematic. Even for timeseries of fixed dimensionality, previous research has found that the Euclidean distance is sometimes unsuitable for real-world applications [31]. Dynamic Time Warping (DTW) is thus preferred to Euclidean distance. Though more costly to compute than the Euclidean distance, DTW aligns two timeseries by considering local distortions and then evaluates the similarity.

DTW does not define a metric space: Though DTW is the most common distance function for timeseries, unfortunately, DTW does not obey the triangular inequality [29]. Hence, techniques that index timeseries based only on *metric* distance values (i.e., metric indices) fail to provide a well-defined search space, and therefore may be unable to be reliably queried.

Locality-Sensitive Hashing (LSH) [32] is a better technique for supporting nearest neighbor approximate similarity search in high-dimensional data. LSH has been shown to be a good approach since traditional indexes do not work well with high-dimensional data due to the curse of dimensionality (in practice datasets with more than twenty dimensions are considered as high-dimensional and difficult to index efficiently [31]). Although it has been demonstrated that LSH scales well with high-dimensional data, the behavior of LSH with data of *variable* dimensionality, such as realistic timeseries of human motion, has yet to be

demonstrated.

this chapter proposes a novel hashing algorithm to index large numbers of realistic timeseries in order to support efficient similarity search. Our algorithm does not guarantee an exact answer to a search, but guarantees that the answer has a high probability of being similar to the query timeseries.

Our contributions are as follows. First, we introduce and formally define the notion of the *general dot product*. Second, we define a hash function using the general dot product. The function hashes timeseries into buckets of multiple hash tables whose total query time is sublinear, i.e., less than $O(log(n))$. Third, we observe that human motion timeseries are not uniformly distributed. Hence, we propose a data-driven hashing function. This approach improves query response times by avoiding skew in the hash function (e.g., by reducing collisons). Finally, we implemented our algorithm and in this chapter report on experiments on real-world datasets, comparing our approach to other approaches. To the best of our knowledge, this is the first work that extends the use of LSH to timeseries.

The rest of this chapter is organized as follows. In Section 3.2 we motivate the need to index realistic timeseries. Section 3.3 reviews the original LSH algorithm. In Section 3.4, we introduce our hashing approach to efficiently retrieve realistic timeseries. Section 3.5 reports on several experiments. Finally, the paper concludes in Section 3.6.

3.2 Indexing Realistic Timeseries

Several papers have provided efficient algorithms to perform queries on timeseries, which are commonly represented as vectors of fixed dimensionality. But some application areas generate timeseries of variable dimensionality. One such area is *streaming motion data*. Motion-related timeseries are usually represented as high-dimensional vectors of varying length because they depend on the duration of a motion. As an actor moves, sensor data is generated as a long stream. Efficiently querying these motion-related streams to find motions of interest is an important task. The motions could be normalized, that is stretched or shrunk, with some loss of information to a fixed dimensionality, but we do not know *a priori* the right dimensionality to normalize to since new motions continue to be

recorded and added to the data store. Furthermore, the normalization loses information by distorting the timeseries to fit a fixed dimensionality. Finally, the continuous processing of new timeseries and queries on existing timeseries may limit other timeseries preprocessing steps such as discretization and dimensionality reduction.

Most of the previous papers on indexing timeseries tackle the problem by either assuming that the timeseries are of the same length or by performing a dimensionality reduction step to normalize the dimensionality of the vectors. Table 4.1 summarizes related research using several dataset features: original dimension, reduced dimension, and normalization method. All of the papers summarized in the table target high-dimensional datasets, but the dimensionality of the data in each case is reduced to speed-up computations in main memory. The intuitive idea is that dimension reduction preserves enough information to quickly discard non-similar timeseries in a search. Then, once a candidate set is identified, the original (non-reduced) timeseries are fetched from disk. One paper of Table 4.1 (Scaled and warped marching [33]) considers more than 50 reduced dimensions during experiments. Unfortunately, the results are only compared to linear search, so it is unclear how indexing high-dimensional timeseries using the scaling method introduced in that paper compares to efficient indexing of timeseries. In any case, all of these papers assume a fixed dimensionality for the timeseries of a dataset and in most of the cases that dimensionality is low, because in practice more than twenty dimensions makes an index inefficient [31].

In contrast to other papers, where timeseries retrieval is tackled by dimension normalization, ours is the first approach that provides a sublinear indexing algorithm explicitly designed to overcome these constraints without preprocessing timeseries or assuming they have similar dimensionality. This approach is especially suitable in scenarios of *stream data processing* such as motion capture, speech recognition, and sensor networks, where timeseries continuously arrive with different lengths, preprocessing steps are not always possible, and low error rates are required.

3.3 Locality Sensitive Hashing

The *Locality-Sensitive Hashing* (LSH) algorithm is based in the idea that if two vectors

Table 3.1. Dimensionalities Considered in Recent Papers on Timeseries Indexing.

Paper (conference)	Original dimension	Reduced dimension	Reduction method
iSAX [34] (KDD08)	480, 960, 1440, 1920	16, 24, 32, 40	iSAX
TS-Tree [31] (EDBT08)	256, 512, 1024	16, 24, 32	PAA
Scaled and Warped Matching [33] (VLDB08)	32, 64, 128, 256, 512, 1024	21, 43, 85, 171, 341, 683	Uniform Scaling
Exact indexing of DTW [29] (VLDB02)	32, 256, 1024	all datasets to 16	PAA

are close together in their original space, then after a scalar projection operation which maps each vector to a point on a line, these two vectors will remain close. If we quantize the line by partitioning it into intervals of same width (*hash buckets*), then we would expect similar vectors to be mapped into the same line interval. The example given below illustrates this idea.

Example 1. *Let $\overrightarrow{p}$ and $\overrightarrow{q}$ be two vectors in R^d and let $\overrightarrow{v}$ be a vector of the same dimensionality, d, as $\overrightarrow{p}$ and $\overrightarrow{q}$. We project $\overrightarrow{p}$ to a number by performing the dot product operation $\overrightarrow{p} \cdot \overrightarrow{v}$. This projection is then quantized into intervals of fixed width w, also known as buckets. The same procedure is repeated for $\overrightarrow{q}$ with the intention that $\overrightarrow{p}$ and $\overrightarrow{q}$ be placed in the same interval, as long as they were similar in their original space (see Figure 3.3). Both the dot product and the quantization operation define the hash function, h, for $\overrightarrow{p}$ with respect to the vector $\overrightarrow{v}$ as follows.*

$$h(\overrightarrow{p}) = \left\lfloor \frac{\overrightarrow{p} \cdot \overrightarrow{v} + b}{w} \right\rfloor \tag{3.1}$$

where $\lfloor \cdot \rfloor$ is the floor operation and b is a random variable taken from the Gaussian distribution $N(0, w^2)$ that helps to evaluate the quantization error.

Hash functions map each high-dimensional vector to a bucket. LSH uses a special

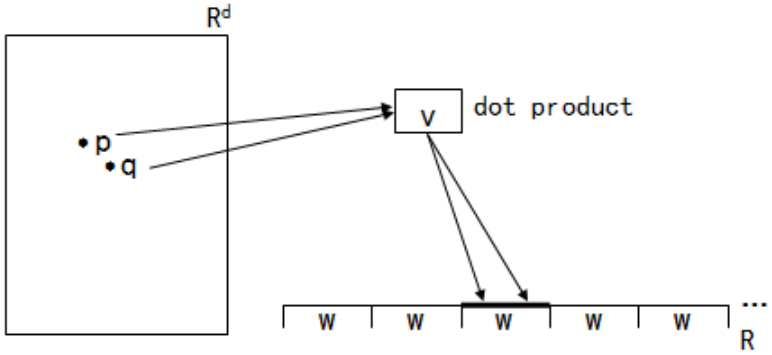


Figure 3.3. Hashing of vectors p and q by scalar projection.

type of hash function called the *Locality-Sensitive hash function*, which is similar to the hash function of Example 1, but with components of $\overrightarrow{v}$ selected at random from a p-stable distribution (Cauchy or Gaussian) [32]. This property makes it possible to statistically guarantee that two similar vectors p and q will map to the same bucket ($h(p) = h(q)$) with high probability. Since LSH is an approximate algorithm, it may not find the exact answer to the query. To reduce the error during query operations, LSH choses more than one hash functions at random and uses each to partition the search space. Additionally, many hash tables are considered in parallel in order to increase the likelihood of finding the right answer in one of the hash tables. This approach has been shown to work well in high-dimensional data, where the density of the space is almost uniform and therefore the selection of a hash functions based on a uniform distribution makes it possible to impose a parametric model to approximate the search space. However, in practice, the distribution of timeseries associated to real world human action such as motion capture, speech recognition, and biometry is not uniform mainly because human actions are constrained by body shape. Hence, the corresponding search space often exhibits a complex structure; this complexity is exacerbated if we consider timeseries of variable dimensionality. Our approach to reduce the complexity of querying this type of search space is detailed in the next section.

3.4 Our Solution

In this section, we propose an index, named *Timeseries Hashing* (TSH), to query realistic timeseries. With this index, we can efficiently find timeseries of any dimensionality in a timeseries database. TSH, like LSH, is based on the idea of projecting similar vectors to the same bucket. But unlike LSH, TSH does not expect a uniform distribution of elements in the search space. Hence, we do not partition the search space using hash functions randomly chosen from a p-stable distribution. Instead, we perform an initial sampling of the data in order to *learn* the best way to partition the search space of a particular dataset. Additionally, we consider the projection of similar realistic timeseries to the same bucket of a hash table via a generalized definition of the scalar projection (dot product).

3.4.1 Locality-Sensitive Hashing for Time Series

Initially, the similarity measure used in LSH was the Hamming distance function between two sequences of bits [35]. Recent papers have explored the idea of using the dot product operation in LSH to compute a scalar value that represents the L_1 or L_2 distance functions between two vectors of the same dimensionality. In this chapter, we extend the definition of the dot product to vectors of different dimensionality. This extension is inspired by the *warping path* generated when computing the DTW algorithm. The warping path is the optimal alignment of two sequences by considering local distortions in the data. To further expand this concept, consider Figure 3.4 as an example. Given two timeseries a and b of different lengths m and n ($n < m$), we first compute a matrix that represents the possible distance values for all the elements in both sequences (Figure 3.4(a)). Then, the warping path is the optimal alignment of two sequences (Figure 3.4(b)), such that the sum of local distances between elements is minimized

A naive evaluation of the warping path is computationally expensive $O(n^2)$, but some heuristics can be applied to approximate its evaluation in $O(n)$. In Section 3.4.1, we discuss the advantages of using these heuristics to generate faster hash functions in TSH. We call the evaluation of the dot product with regard of the warping path as the *general dot product*, and formally define it as follows.

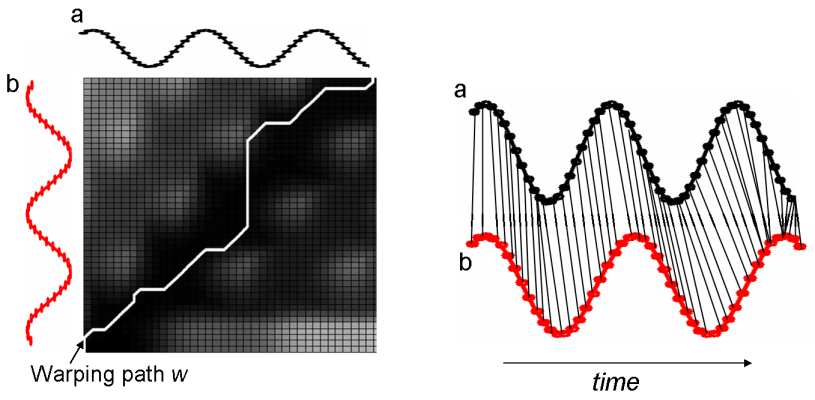


Figure 3.4. (a) The warping path of two timeseries represents the path with minimal distortion in the distance matrix. (b) The local alignment is basically the matching of two timeseries by considering local distortions in the time axis.

Definition 3 (General Dot Product). *Given timeseries* $a = [a_1, a_2, \ldots, a_n]$ *and* $b = [b_1, b_2, \ldots, b_m]$ *and an optimal alignment function*

$$f(a) = [f(a_1), f(a_2), \ldots, f(a_n)] = [b_1, b_2, \ldots, b_m]$$

which matches each element from a *to* b *with respect to the warping path, the general dot product is defined as:*

$$\begin{aligned} a \odot b &= \textstyle\sum_{i=1}^{n} a_i f(a_i) \\ &= a_1 f(a_1) + a_2 f(a_2) + \ldots + a_n f(a_n) \end{aligned}$$

Note that the original definition of the dot product is a special case of the general dot product. While the dot product performs a pair-wise comparison of two vectors that have the same dimension, in the general dot product the comparison is not necessarily pair-wise since the two timeseries may not be linearly aligned. The general dot product can perform the scalar projection of timeseries with different dimensionality.

We use Definition 3 to take pairs of non-linearly aligned elements of two timeseries in order to project the similarity of both timeseries onto a line as follows. We embed Definition 3 into a *locality-sensitive hash function* based on scalar projections to index

vectors of variable dimensionality with respect to a randomized vector $\overrightarrow{v}$. The new locality-sensitive hashing function for a vector $\overrightarrow{p}$ is defined as follows.

$$h(\overrightarrow{p}) = \left\lfloor \frac{\overrightarrow{p} \odot \overrightarrow{v} + b}{w} \right\rfloor \tag{3.2}$$

Good query performance using TSH depends on a careful choice of parameters such as the dimensionality of $\overrightarrow{v}$, the number of hash functions, and the number of hash tables. We detail these tuning parameters in the next section.

Parameters

While different hash functions partition the search space with more detail, more than one hash table increases the probability of retrieving the right timeseries in at least one of the tables. Additionally, the dimensionality of vector $\overrightarrow{v}$ also affects the probability of collision(s).

- $|\overrightarrow{v}|$: $\overrightarrow{v}$ is the vector that characterizes the projection performed by a hash function and $|\overrightarrow{v}|$ is the dimensionality of that vector. We empirically determined that if this value is constant, the number of collisions in the hash table will dramatically increase. In contrast to the original LSH, the dimensionality of $\overrightarrow{v}$ is not constant rather it is the dimensionality of a vector randomly chosen from the data during initialization. More details on this process are provided in Section 3.5.1 which describes the way how we choose the parameters that optimize the use of the index.

- $\boldsymbol{k}$: indicates the number of hash functions in a hash table. If we consider random values for each vector, we will obtain independent hash functions since their components are randomly chosen from a Gaussian distribution $N(0, 1)$. This property, although desirable in high-dimensional and artificial datasets, is not always efficient when the structure of the data is complex or forms clusters.

- $\boldsymbol{l}$: represents the number of hash tables in LSH. By concatenating l hash tables, we reduce the probability of reporting a wrong answer for retrieving the closest element

to the query timeseries.

Non-uniform partitioning of the search space

Since it may be difficult to analyze a search space of variable dimensionality, we instead study the distribution of elements projected in the buckets of one hash table to decide the goodness of using an approach based on scalar projections. The hash functions described by Equation (3.2) are locality sensitive since the component values of vector $\overrightarrow{v}$, which define the hash function, are chosen at random from a p-stable distribution, although their dimensionality is also chosen at random. The distribution of timeseries into buckets shows that the random projection operation still produces collisions in the datasets considered in this chapter. Large numbers of collisions negatively affects the performance of the hash tables since many elements are considered as candidates to solve a similarity query. We would like the timeseries to be more uniformly distributed among the buckets to obtain sublinear times during queries.

Collisions in hash tables can be explained by the way the hash functions discretize the search space into buckets. The original LSH algorithm randomly tessellates the search space with k hash functions chosen at random from a Gaussian distribution, as shown in Figure 3.5(a). By increasing the value of k, timeseries are uniformly distributed into buckets, but this decreases the accuracy since it is more likely that similar vectors fall into different buckets. Recent papers have also noticed the same problem with high-dimensional datasets in R^d. For example, in [36] the buckets with collisions above a certain threshold are re-partitioned to reduce skew in the hash tables, as shown in Figure 3.5(b). This approach however leads to a hierarchy of hash tables which is difficult to scale when new elements are indexed. This is because the structure of different levels of buckets will change as some buckets become denser than others during insertion of new vectors.

Our approach to reducing the collisions that occur in the hashing of realistic timeseries with the general scalar projection introduced in this chapter is to tessellate the search space with timeseries taken from an initial sampling of the the dataset. This results in fine-grained partitions only in dense regions and wide partitions in sparse regions, as shown in

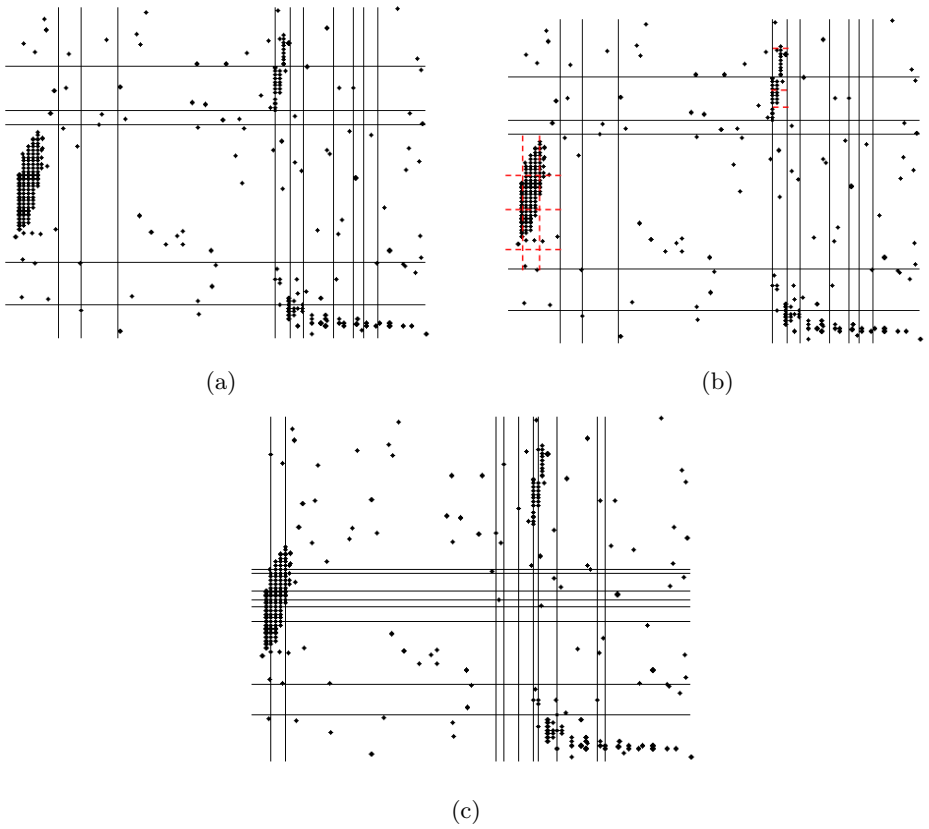


Figure 3.5. (a) Random partitioning of the search space (b) Hierarchical partitioning of the search space (c) Partitioning of the search space by considering elements from the space itself.

Figure 3.5(c). We repeat this process for the l hash tables in order to perform a query with different resolutions at each hash table. Intuitively, a query is solved by quickly hashing the query object into each hash table and then joining the results, as shown in Figure 3.6. Note that, as in the original LSH, the candidates that solve the query become restricted to a query region of non-arbitrary shape with few elements inside. Care should be taken to sample the diverse types of timeseries expected since the data in streaming systems may slightly change over time. If trends are detected in newly arrived data that lead to a significant increase in the number of collisions, the dataset should be re-sampled and a new hash function constructed to re-hash the data.

Queries on TSH

It has been shown that the DTW similarity measure does not describe a metric space since the triangle inequality property does not hold. Therefore, its iterative use to organize timeseries based on distances will eventually degrade search accuracy. The hash functions of TSH directly map each timeseries into one bucket and avoid partially exploring the index to find the closest timeseries through successive comparisons between pairs of timeseries. Hence, TSH guarantees to evaluate fewer distance comparisons to perform a query. Moreover, timeseries do not need to be fully stored in main memory. Instead, we store in main memory the index of the bucket as a key and the name of the timeseries file in hard disk as a value. By doing this, we reduce the main memory requirements in TSH.

The worst-case time complexity of retrieving a timeseries in TSH is $O(kl/t/+n)$ where k is the number of hash functions, l is the number of hash tables, and $/t/$ indicates the time spent to evaluate the general dot product between two sequences. Finally, the variable n represents the number of candidates obtained by concatenating results at each hash table; a large value of n indicates a large number of collisions in the hash tables. Recent papers on LSH show that k and l are sometimes too large to ensure that the error value is low. A large value of k is especially problematic because it duplicates the entire dataset in memory. The partitioning of the search space using timeseries chosen from the dataset yields a more efficient tessellation as shown in Figure 3.5. This data-driven approach improves the accuracy for queries with few hash tables. Low values of k and l make the complexity analysis of a query mostly dependent on the cost of computing the general dot product (described in Definition 3) and the number of candidates retrieved. While we have introduced two techniques to reduce the value of n (variable dimensionality in hash functions and non-uniform discretization of the search space), techniques to reduce the value of $/t/$ are discussed in the next section. Note that $O(kl/t/+n) << O(N)$ commonly holds even for hash tables with moderate collisions, where N is the size of the timeseries dataset.

Reducing the Hashing Cost

The general dot product definition uses the warping path generated by the Dynamic

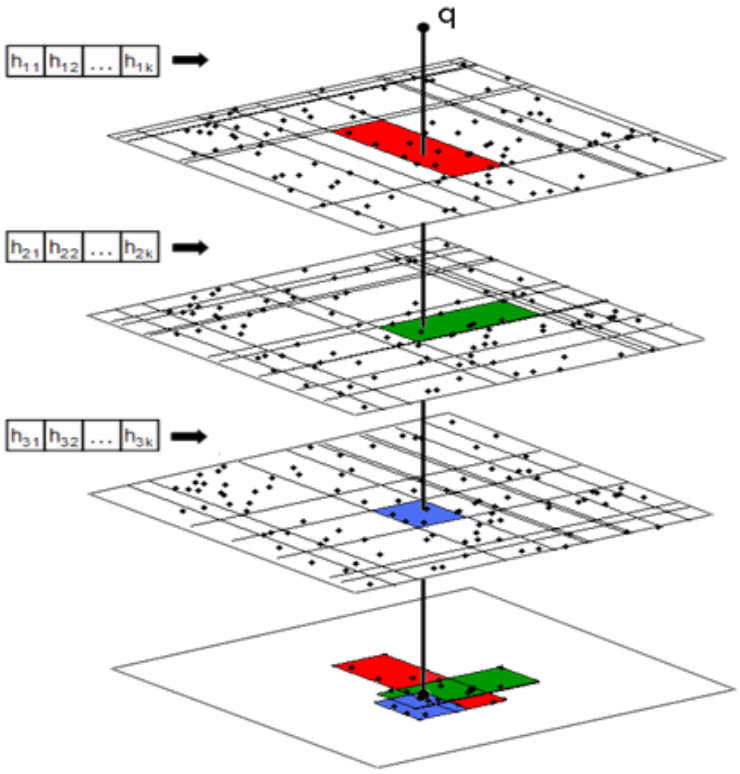


Figure 3.6. Three hash tables, each one discretizing the same search space with different hash functions.

Time Warping algorithm to find a non-linear alignment between two timeseries before projecting their similarity into one scalar value. However, this algorithm has a quadratic time complexity that limits its use to only timeseries of short length. Previous papers have provided some techniques to improve the speed of compute it without significantly degrading the correct alignment of two timeseries. As an example, consider the *Sakoe-Chiba band* and the *Itakura parallelogram* shown in Figure 3.7(b) and Figure 3.7(c). The enclosed areas correspond to positions within the distance matrix where the warping path is expected to lie. Hence, positions located out of these bands do not need to be evaluated. By evaluating the warping path within a constrained band, we do not need to compute all the values within the matrix. The width of each band is specified by an external parameter. However, if the optimal warp path does not completely fall inside the band, some error will be introduced during the total alignment. For the indexing of timeseries in TSH, we will use the Sakoe-

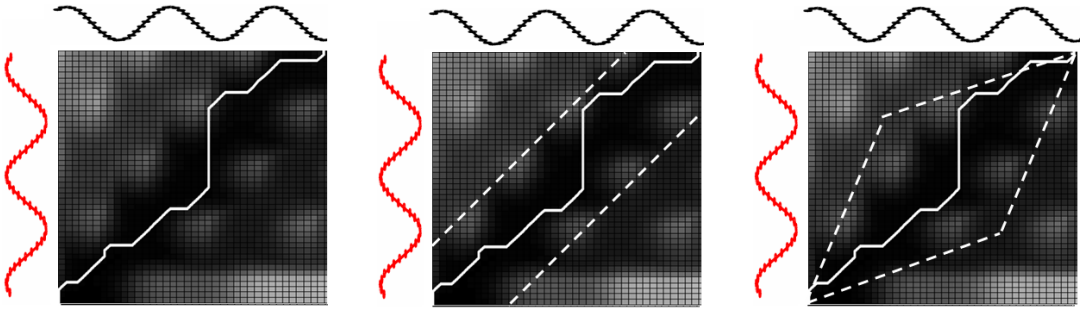


Figure 3.7. (a) Original DTW alignment (b) Sakoe-Chuba Band constraint (c) Itakura Parallelogram constraint. Both (b) and (c) have a width of 10 and 22, respectively.

Chiba band to speed-up the alignment of two timeseries, yielding faster hash functions for realistic timeseries.

3.5 Evaluation

We performed a comprehensive performance evaluation of our algorithm in terms of *precision*, *response time*, and *scalability* against the following indexes.

***LSH* [32]** — This is the original implementation of the hashing scheme extended in this chapter. A dimensionality normalization step was performed on the data in order to be able to index timeseries, since they had to be of the same dimensionality to use LSH, with Euclidean distance.

***TS-tree* [31]** — This data structure was specially designed to index timeseries by avoiding subtree overlap through lexicographic ordering on time series. Timeseries are normalized and quantized into symbols to obtain a compact description. This feature makes TS-tree faster than region-based indexes like R-tree.

***R-tree* [37]** — This spatial index is commonly used during experiments on timeseries in the literature [29, 31, 33]. We use the Euclidean distance to evaluate dissimilarity on normalized timeseries.

M-tree **[38]** — This metric index uses a distance function to organize objects based on dissimilarity. Special nodes are created to serve as pivots and objects are stored in leaves. Since DTW is not a metric distance, we expect less accurate results in this index due to the lack of a metric search space. We use DTW in M-tree in order to have other index that retrieves variable-dimensional timeseries without normalization.

All of the experiments were performed on a 3.6 GHz Pentium 4 with 2 GB RAM. In this section, we describe those experiments and their results on the following datasets.

1. ***CMU motion capture database***: This dataset is considered the largest motion capture dataset publicly available. It consists of 2,435 motion clips, each one with segments manually labeled, performed by 83 actors, and with 26 hours of motion in total. The timeseries are generated by tracking the Y-axis position of the actor's right hand as done in [39]. The dimensionality of this dataset ranges between 53 and 5,237.

2. ***Mocap Database HDM05***: This dataset contains 70 motion classes, each with 10 to 50 different realizations performed by five actors. In total there are 1,500 motion clips and 210 minutes of motion data. From each motion clip we generate one timeseries by tracking the right hand as with the CMU dataset. The dimensionality of this dataset ranges between 49 and 4,034.

3. ***Motion from video***: We generate this dataset by extracting timeseries from video data by following the approach discussed in [40]. The videos were taken from the Action Database [22] and consist of 6 different types of motions (boxing, hand clapping, hand waving, jogging, running, and walking) performed by 19 actors. The number of frames in the videos varies from 300 to 750 frames and includes 40 minutes of video in total. The dimensionality of this dataset ranges between 10 and 58.

3.5.1 Experiment 1: Selection of parameters

As discussed before, TSH reports efficient results when adequate values for k, l, and $/\overrightarrow{v}/$ are chosen. Hence, we propose here an effective method to choose a suitable combination of these parameters constrained by the response time and precision. Intuitively, we

are looking for a method which finds the number of hash tables (l) and hashing functions (k) which report the most precise and fastest results for a certain dataset. To reach that goal, we generate different instances of TSH for $k = \{8, 16, 32, 64, 128\}$ and $l = \{1, ..., 20\}$ and then measure the corresponding response time and precision for all the datasets.

Let $1 - \varphi$ and t be the *error degree* and *response time* of a query. The value of φ is obtained by computing the precision of a 1-nearest neighbor query: the number of times that the set of elements retrieved by TSH also contains the right answer obtained by a brute force algorithm for the same query. If after 100 queries, TSH always reports the right timeseries, then the error degree will become zero. The response time is the time spent to evaluate φ divided by the number of queries performed. Unless stated otherwise, we follow the above definitions of precision and response time in all our experiments.

First, we find the number of hash functions k that reduces the error degree such that $1 - \varphi$ is less than a threshold ($1 - \varphi < 0.03$). As shown in Figure 3.8(a), the error degree for the CMU motion dataset (each individual line) is reduced when more hash tables are considered. In that figure each k value is associated to one l value, and both define a particular instance of LSH that considers the error $1 - \varphi$ below the threshold. Once we find different combinations for k and l, we evaluate the dependence of these values with respect to time $T(k, l)$ such that the minimum value of $T(k, l)$ is chosen. This process is outlined in Figure 3.8(b). Each peak represents an suboptimal combination of k and l obtained from Figure 3.8(a). We are looking for the best of these combinations that reduces the response time. Variable $/\overrightarrow{v}/$ takes the dimensionality of a vector randomly chosen from the dataset during the non-uniform partitioning of the search space. Since the timeseries are of variable length, the value of $/\overrightarrow{v}/$ is a non-constant number in the index. For constant values of $/\overrightarrow{v}/$, we observed an increment in the number of collisions for timeseries of variable length. The same approach is performed in the other two datasets to find a suitable combination of k and l which are used in the precision, response time, and scalability experiments described next.

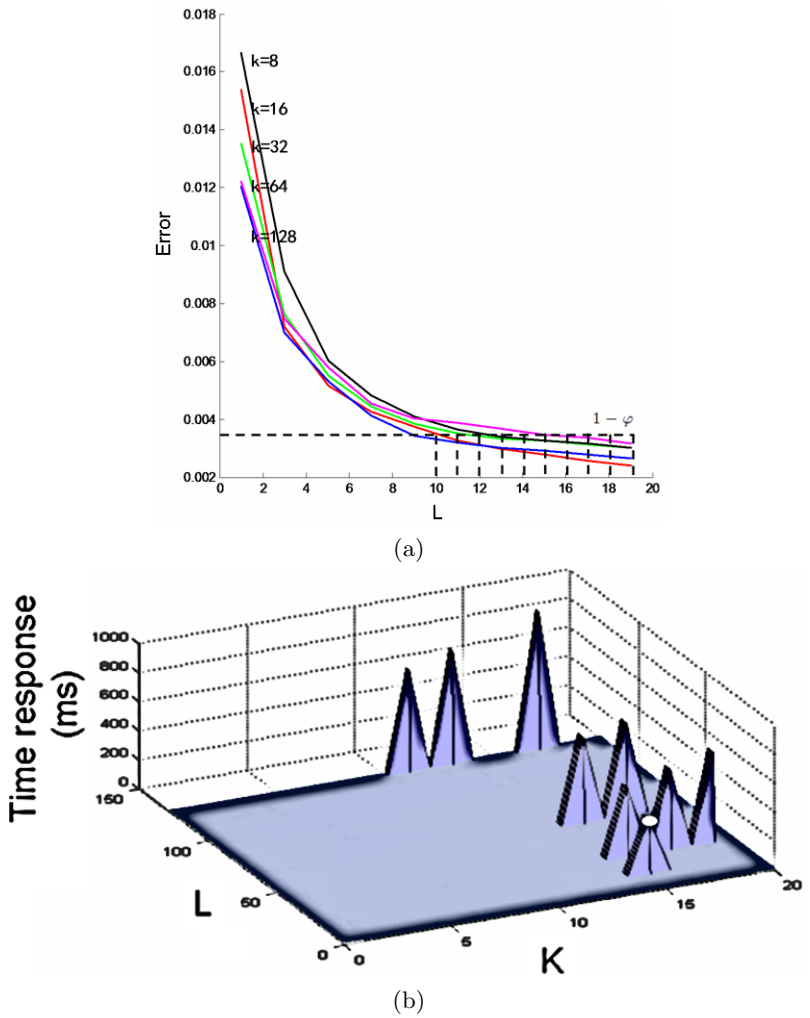
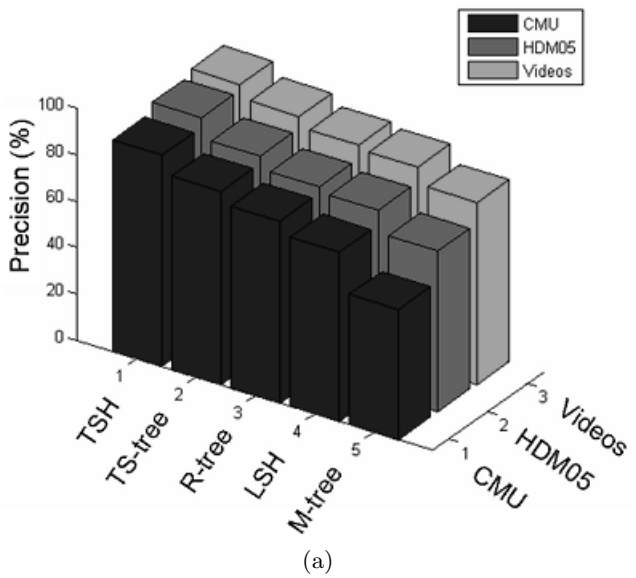


Figure 3.8. (a) Error values for different combinations of number of hash functions and hash tables. Note that for a given error threshold $1 - \varphi$, we find candidate pairs of k and l that minimize the *error values*. (b) These combinations are then refined to choose the values that optimize the query in terms of *response time*.

3.5.2 Experiment 2: Precision

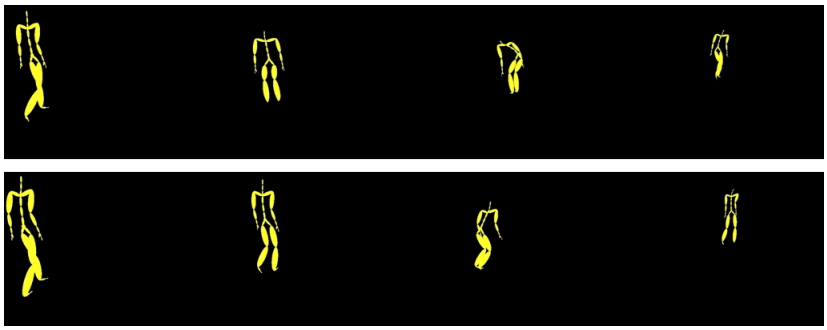
The aim of this experiment is to evaluate the average precision of our approach against other well-known indexes. Given an index i and a database d, we evaluate the average precision by splitting the dataset into two parts: 90% for indexing and 10% for testing. Once we index the dataset, a random timeseries from the testing part is chosen for searching. We perform a 1-nearest neighbor query and check if it includes the same element reported



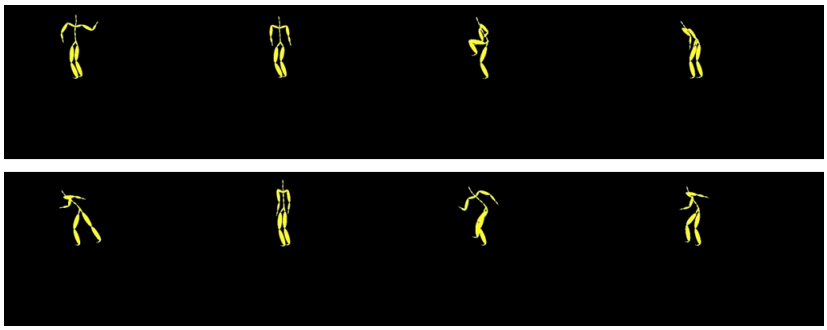
(a)

Figure 3.9. Average precision for the three datasets in decreasing order.

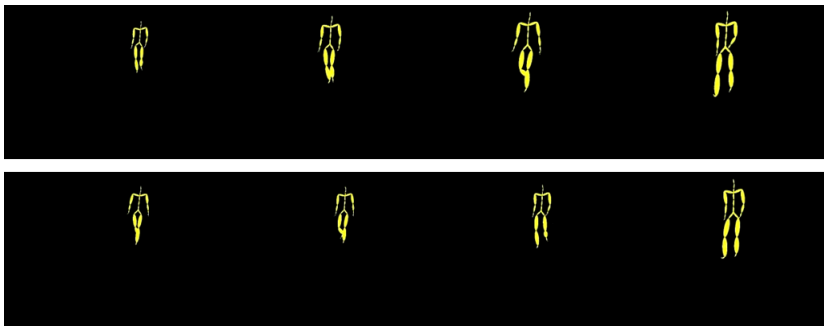
by a linear scan with DTW as dissimilarity function. After repeating this process for 100 iterations, we average the precision values of the index i in order to avoid some biased results during the experiment. Figure 3.9 condenses the average precision values sorted in decreasing order. TSH performs better than other indexes independent of the size of the dataset. Note that M-tree reported the worst results in all the cases. This is expected since the DTW distance function does not fully define a suitable metric space to perform similarity queries in M-tree. The length normalization of timeseries and the use of Euclidean distance as similarity measure make R-tree a better choice than M-tree in terms of precision. We see that the original LSH algorithm with Euclidean distance does not perform as well as TSH, especially in the largest dataset (CMU). This is because TSH does not need to normalize the timeseries in the dataset to the same dimension and because TSH uses DTW as its similarity measure. TSH thus takes advantage of the original information contained in the data to distinguish non-similar timeseries. Figure 3.10 visually shows examples of 1-nearest neighbor queries in TSH.



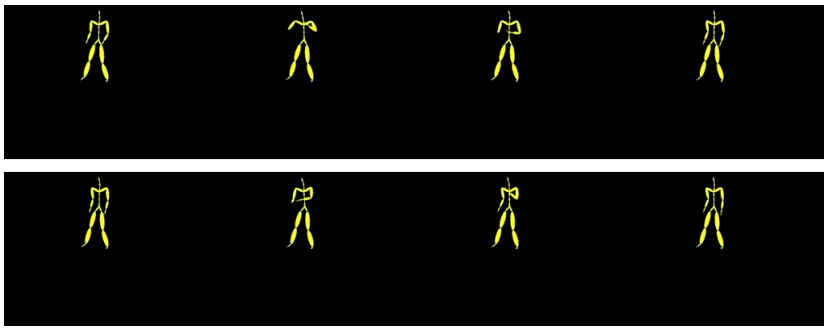
(a) Crouching



(b) Kicking and punching

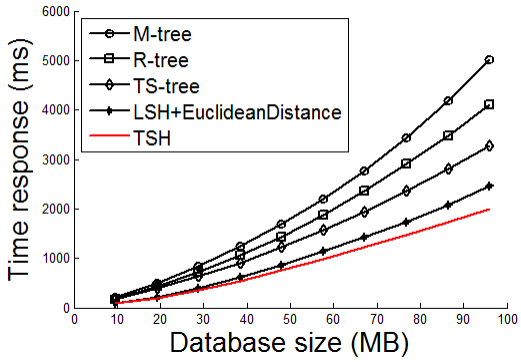


(c) Walking

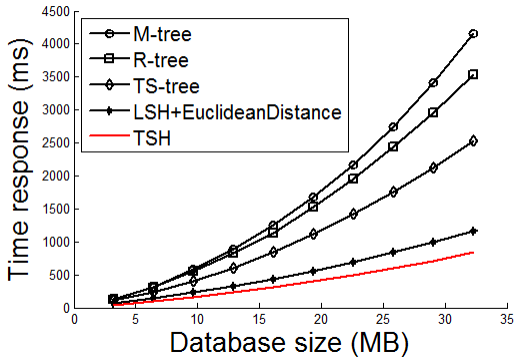


(d) Rotating arms

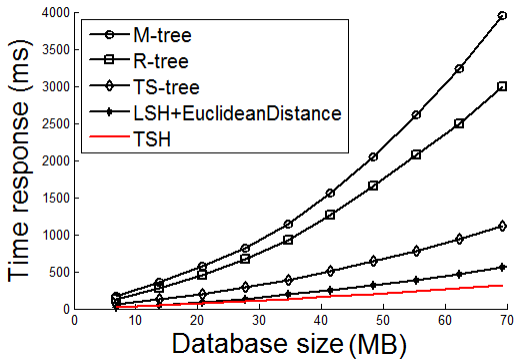
Figure 3.10. Results of four 1-nearest neighbor queries in TSH for the HDM05 database. While the first row of each figure represents a query, the second row shows the closest movement found in the database.



(a)



(b)



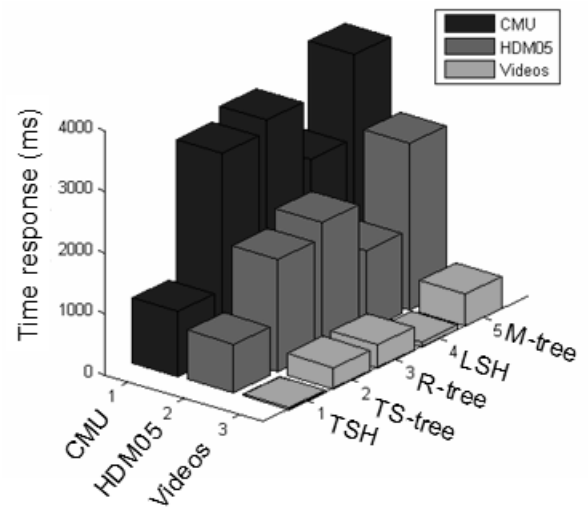
(c)

Figure 3.11. Scaling measurements for (a) CMU (b) HDM05 (c) Videos datasets

3.5.3 Experiment 3: Response time

The goal of this experiment is to measure the average time spent to retrieve the nearest timeseries from a dataset. We perform a similar procedure to that of Experiment 1 to evaluate the average response time and show the results in Figure 3.12. Although the hash-

based approaches (TSH and LSH) have are fastest, we already observed from Experiment 1 that TSH reports more precise values than LSH. TS-tree exhibits acceptable precision and speed in all of the datasets. However, TSH is still faster and more precise than TS-tree. These results indicate that TSH is more efficient than any index considered in this chapter.



(a)

Figure 3.12. Average response time for the three datasets. The order of indexes is the same as in Figure 3.9.

The alignment of two time series with different dimensions may be costly, especially in the case of motion sequence where time series are expected to be large. Every time we query a time series in the database, TSH evaluates the DTW distance between that time series and the vector v of every hash function to find similar time series. The evaluation of the DTW function may hinder the performance of the index, particularly when several hash tables are considered in parallel. As shown in Table 3.2, the use of the Sakoe-Chiba band alleviates the query process of realistic time series for the datasets considered in this chapter. This is because the computational order of the DTW algorithm is reduced by considering a global constraint in the evaluation of the distance matrix of two large time series.

Table 3.2. Use of the Sakoe-Chiba Band to Speed Up Similarity Queries on the Datasets Considered in this Chapter. Each Value Represents the Average Response Time of Similarity Queries.

Dataset	$\text{TSH}_{Sakoe-Chiba}$ (ms)	$\text{TSH}_{no\ Sakoe-Chiba}$ (ms)
CMU	1062.52	1581.31
HDM05	803.31	1148.82
Video	35.26	48.62

3.5.4 Experiment 4: Scalability

In this experiment, we want to study the behavior of TSH as the size of the database increases. We vary the size of the database and measure performance to determine scalability (the performance should be (at worst) linear in the size of the data in order to ensure good performance for large databases). We incrementally increased the size of the dataset from 10% to 100% of the original size (by selecting a random testing subset from the dataset). We measured an average response time, as was done in previous experiments. Figure 3.11 shows that for the datasets considered in this chapter, TSH exhibits good scalability. In all the cases LSH is the closest contender. This is because both algorithms are good at distributing timeseries into buckets uniformly and quickly retrieving them using hashing functions. However, in contrast to LSH, TSH is specially designed to reduce the number of collisions for timeseries of variable length.

3.6 Conclusion

In this chapter, we generalized the scalar projection (dot product) to support the indexing of timeseries of variable length using a novel hashing technique. We proposed one approach to reduce the collisions associated to map timeseries into buckets of a hash table. This approach enables us to define multiple hash tables that increase the probability of retrieving the nearest neighbor of a query timeseries in sublinear computational time. We conducted performance studies on three real datasets associated to human motion from different sources (systems of motion capture and video data). One reason for the popularity of LSH is that, in theory, it ensures that that two similar vectors are mapped similarly.

However, much of the theoretical analysis provided in the original implementation of LSH relies on statistical assumptions like a uniform distribution of the data. As we saw in this chapter, this is not the case for motion datasets. Rather than providing a theoretical analysis, we empirically look for the combinations of parameters that yield low error rates for similarity queries. This data-driven approach to constructing hash functions is how we efficiently index human motion timeseries data that come from non-uniform distributions. Our results show that the approach introduced in this chapter can retrieve timeseries of non-arbitrary dimensionality efficiently. In particular, TSH is superior for large datasets with large and variable dimensionality, the hardest case when the indexing of timeseries associated to human motion is considered.

CHAPTER 4
DEMONSTRATION OF TIMESERIES SENSITIVE HASHING FOR HUMAN MOTION

4.1 Introduction

The goal of the research presented in this demonstration is to make it easier for applications to index and query timeseries that are of high and variable dimensionality. These timeseries are found in many application areas, such as image and video processing, data stream mining, data compression, and near duplicate detection. For instance, a timeseries of variable dimensionality is generated when a process (e.g., the trajectory of a person walking in a video) is measured over uniform time intervals, but with no predetermined length (e.g., not every video or activity within a video has the same duration). Previous research has developed efficient algorithms to index and query timeseries that are vectors of fixed dimensionality. Current practice is to normalize variable dimensionality timeseries to some fixed dimensionality. But when processing a live video stream, for instance, new timeseries are continually being produced and the *a priori* (often incorrect) choice of a single, fixed dimensionality may lead to important data in a timeseries being manufactured or discarded. It would be better to index each timeseries without normalizing it.

Table 4.1 summarizes a few of the key papers in previous research. All of the papers listed target high-dimensional datasets. Each chooses to normalize by reducing the dimensionality of the data to some fixed dimensionality, and in most of the cases that dimensionality is relatively small.

In this demonstration, we will show how to index timeseries without normalizing or reducing dimensionality using an indexing technique called Timeseries Sensitive Hashing (TSH) [4]. In contrast to other approaches, TSH has linear or sublinear time complexity

Table 4.1. High-dimensional Datasets and Reduction Techniques Considered in Recent Papers on Timeseries Indexing.

Paper (conference)	Original dimension	Reduced dimension	Reduction method
iSAX [34] (KDD08)	480, 960, 1440, 1920	16, 24, 32, 40	iSAX
TS-Tree [31] (EDBT08)	256, 512, 1024	16, 24, 32	PAA
Scaled and Warped Matching [33] (VLDB08)	32, 64, 128, 256, 512, 1024	21, 43, 85, 171, 341, 683	Uniform Scaling
Exact indexing of DTW [29] (VLDB02)	32, 256, 1024	all datasets to 16	PAA

for both indexing and searching without reducing the timeseries to a fixed dimensionality. As we will demonstrate, TSH is especially suited to *stream data processing* such as motion capture, real-time video analysis, speech recognition, and sensor networks, where timeseries with different lengths continuously arrive, preprocessing steps are not always possible, one-scan algorithms are needed, and low error rates are necessary.

4.2 The Demonstration of TSH

This section gives a short tutorial on TSH. The demonstration will use a web application to take the audience through a series of multimedia queries of increasing complexity, from image to motion capture and video data. The web application allows a user to perform four main activities: 1) initialize the interface with a random sampling of objects chosen from a database, 2) report how elements are distributed in hash buckets, 3) visualize a hierarchical clustering of the entire dataset, and 4) perform a similarity search. We invite readers to visit the TSH project website to experiment with TSH in the on-line demo and download the Java implementation[1]. The hash tables are persistently stored using BerkleyDB [41], so the software can scale to support large datasets (i.e., it is not limited by available memory).

[1] http://omarflorez.info/index.php?id=tsh

(a)

(b)

Figure 4.1. (a) List of primate skulls randomly selected from the database. (b) A similarity query in the system. The first element represents the query (distance of 0) and repeated elements found in parallel hash tables were discarded.

4.2.1 A Quick Overview of TSH

TSH is an index that clusters timeseries with similar shape which are close in the $\Re^1$ space. TSH employs the concept of *general dot product* to compare two vectors of different dimensionality using a non-linear alignment of elements, rather than using the Euclidean norm in the $\Re^n$ space.

TSH embeds general dot product into a number of hash functions, which allows each timeseries to be projected into $\Re^1$ (e.g., a bucket) with high probability of finding similar

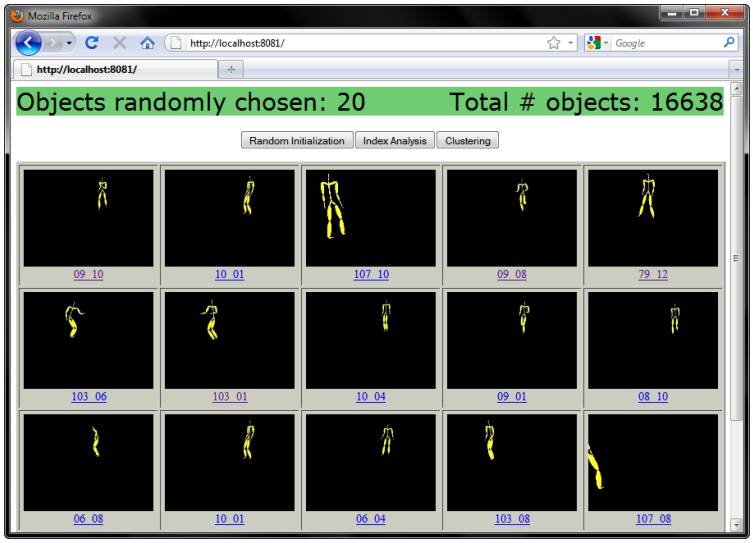
elements in the same bucket. TSH chooses timeseries from the database to construct the hash functions, which we previously observed did improve precision [4]. Additionally, TSH uses multiple hash tables, performs a search by hashing within each table, and unions the partial results. In summary, TSH is formed of L hash tables, each one defined by K hash functions that project elements to buckets of a chosen width in $\Re^1$ (the width is a parameter that can be used to tune TSH).

4.2.2 Example Searches using TSH

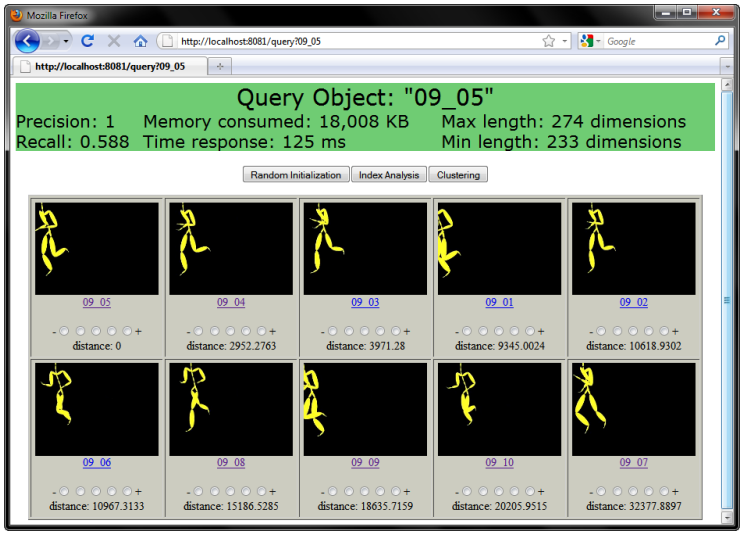
Figure 4.1 (a) shows a random sampling of a *Primate Skulls* dataset. We generate a timeseries for each image by computing its centroid and then evaluating distances from that point to the pixels in the border of the skull. The resulting timeseries is expressive enough to distinguish important features such as nose angle, depth of the lower jaw, and prominent forehead.

Figure 4.1 (b) shows a screenshot of the application when a user clicks on an image performing a query of an adult female *Sooty Mangabeys* primate. This primate lives on the west coast of Africa and belongs to the *Cercopithecidae* family and *Catarrhini* infraorder. While 80% of the neighbors retrieved by TSH (a, b, c, d, e, and g) belong to the *Catarrhini* infraorder and share important features such as long fangs, similar forehead, and very convex napes, the remaining 20% (f and g) also exhibits those visual features, but they belong to the *Platyrrini* infraorder. Both *Catarrhini* and *Platyrrini* are two close instances of five possible infraorders for primates. This example shows that timeseries can be used to evaluate similarity queries for image data. The title of the application also contains relevant information about the query such as *precision/recall* of the subset retrieved and the *time/memory* spent during the query in the system. Additionally, the demo reports variance in the dimensionality (*min/max length*) associated to the result set.

In the above example, we took advantage of the same orientation and uniform background to extract well defined timeseries and obtain consistent results in the *Primate Skull* database. However, the timeseries extracted from motion capture data are more challenging and complex to analyze since the motion performed by different actors can be in any



(a)



(b)

Figure 4.2. (a) Motion capture videos chosen at random. (b) A similarity query performed by TSH. Note that all the videos retrieved belong to the same running activity.

direction, be composed of other small movements, and have a larger variance in duration, as shown in Figure 4.2 (a). The timeseries defined for each instance of the database represents the position of the right hand of each actor in the z-axis. The result of querying the running activity *09_05* is illustrated in Figure 4.2 (b). It shows the retrieval of ten videos, each one containing different actors doing the activity *09*. The *recall* is lower in this case (58.8%)

than in Figure 4.1 (b) since more than ten elements belong to this class, but TSH only retrieves a subset of them. In other words, TSH only retrieves the timeseries projected to the same buckets in multiple hash tables with high *precision* (100%). The user can obtain higher *recall* values by increasing the bucket width, but at the cost of increasing the response time.

Additionally, we show search effectiveness with a subset of 100,000 elements of *The Million Song Dataset,*[2] a freely-available collection of audio features and metadata. TSH will provide consistent results if the result set obtained by a 10-NN query is included in the one retrieved by a linear scan, a brute force algorithm that yields complete precision. Figure 4.3 reports average precision of TSH, with respect to a sequential search, which ranges between 92% and 95% over different database sizes.

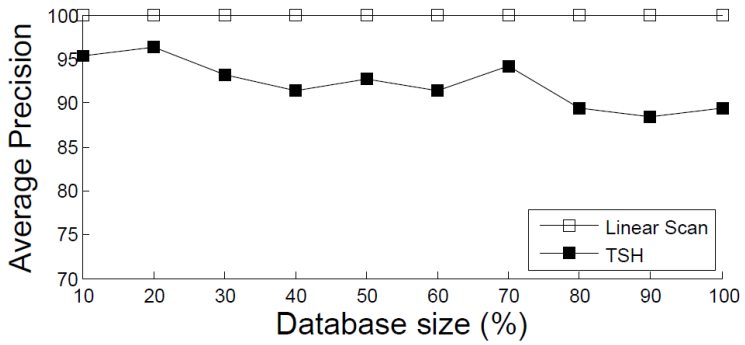


Figure 4.3. Average precision for TSH.

4.2.3 Clustering

The demo will also allow users to explore the notion of using a general dot product to create a hierarchical clustering. TSH projects similar timeseries to the same buckets in multiple hash tables, forming a particular neighborhood in the projected space. Those points that have more than T elements and are within a distance of D are denominated as *core points*.

[2] http://labrosa.ee.columbia.edu/millionsong/

TSH aggregates clusters into increasingly larger clusters in a hierarchy as follows. Pairs of clusters that have the shortest distance between their members can be connected. Those timeseries that are not *core points* can be considered as outliers and safely removed. The algorithm stops when a single cluster is reached. Otherwise, we increase the value of D to merge distant clusters. TSH reduces the time complexity of the original hierarchical linkage clustering algorithm from $O(n\log n)$ to $O(n)$ because of its constant response time for nearest neighbor queries.

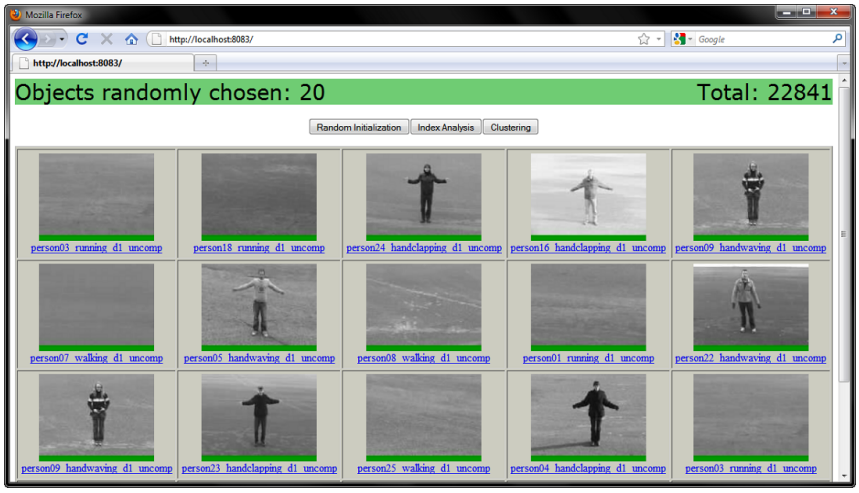
Figure 4.4 (a) shows a random selection from the *Motion from Video* dataset. It contains six types of activities performed by 19 actors with dimensionality varying from 112 to 184 dimensions. A timeseries is extracted from each video by defining intensity gradients and then evaluating their spatial distribution over time. The clustering on this data with TSH, as shown in Figure 4.4 (b), reports some relevant characteristics underlying the dataset.

First, note that five groups were discovered, each one shown in a different color. Each group is a type of motion performed by an actor. While the cluster in red (a) merges two types of similar activities (jogging and running), the other clusters shows a proportional distribution of elements. Next, the highest link in the display shows the existence of two large groups ($\{a, b, c\}$ and $\{d, e\}$). In practice, those groups correspond to *hand-related* (hand waving, hand clapping, and boxing) and *feet-related* (walking, jogging, and running) activities. While clustering is approximated with TSH, it quickly groups elements projected similarly in $\Re^1$ and uses the general dot product to find similar timeseries from video. Those timeseries are often more challenging since they contain noise related to illumination, shadows, and camera motion.

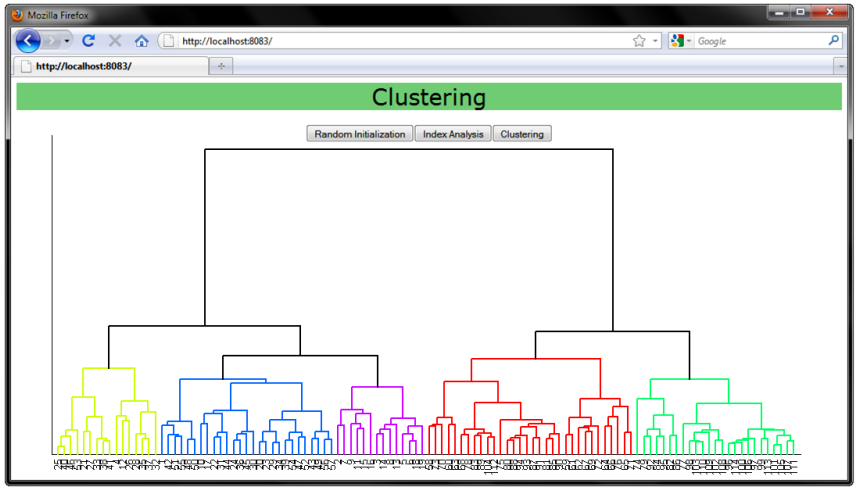
4.3 Architecture

Our implementation builds on the Jetty HTTP server and Java Servlet container, which provides both dynamic content interaction with multimedia databases and the TSH index.

Before we can execute similarity queries on objects, we need to find its timeseries representation in the *Data* layer. The technique to extract those features from an object



(a)



(b)

Figure 4.4. (a) A random selection from the *Motion from Video* database. (b) Hierarchical clustering performed by TSH. Note that timeseries extracted from videos are the most complex shown in this demonstration.

will vary according to the nature of its underlying data (images, music, video, motion capture, etc). In our running examples, we showed multimedia objects to the user, each one linked to a timeseries in the *Presentation* layer. Those timeseries are encoded in $< key, value >$ pairs within TSH; *key* is an integer representing a bucket id in TSH and *value* is a string representing the path to the file containing the timeseries pattern on hard-disk. Thus, mappings allow us to save space by not storing the entire timeseries in main

memory. The index will perform similarity queries and clustering by quickly obtaining keys in multiple tables and collecting the timeseries found into a single result set in the *Structure* layer. Eventually, TSH will contain more mappings than permitted in main memory. TSH supports the insertion of large multimedia databases with fast lookup operations in TSH by extending the HashMap representation of BerkeleyDB to thus enable persistence in the *Persistence* layer.

4.4 Previous Work

Previous research on efficiently indexing timeseries typically relies on dimensionality reduction to eliminate variable dimensionality. As discussed in Section 4.1, current practice is to employ on dimensionality reduction techniques (iSAX [34], PAA [31], Uniform Scaling [29], etc.) to make timeseries more tractable to compute. These indexing techniques fall into two categories: region-based and lexicographic. R-tree is a region-based index commonly used for experiments with timeseries. Its main downside is the presence of overlap between regions for high-dimensional data. One example of a lexicographic index is TS-tree [31], which avoids subtree overlap during insertion. There are (at least) two clustering techniques that employ a hashing approach [42, 43]. However, they only consider the case where points have the same dimensionality and Euclidean distance is the only option as a similarity measure.

4.5 Summary

The demonstration will guide the audience through examples in a new index called TSH. We demonstrate the potential of TSH to perform quick analysis of complex timeseries in multimedia data. Particularly, queries in TSH show high precision and visual soundness and the hierarchical clustering seems to group similar elements coherently. The audience can download our Java implementation from the project's website and try their own queries during the demonstration.

CHAPTER 5
RULE EXTRACTION TO EXPLAIN VEHICLE INTERACTIONS IN VIDEO WITH GUARANTEED ERROR VALUE

5.1 Introduction

In many real-world applications, data takes the form of an ordered sequence of items that arrive continuously. Over time, a huge amount of data can accumulate and the distribution of data within a stream can vary. Traditional examples include Sensor Networks [44] and Internet Packet Streams [45]. These scenarios provide explicit representation of atomic elements and the frequency of their combinations must be computed in a single scan because of the continuous arrival of data.

Video can also be modeled as a data stream. Video is widely used in real-time monitoring applications, e.g., of an oil spill, a store entrance, or an airport. In a video stream we are interested in discovering and monitoring the hidden rules that govern the behavior of multiple objects occurring in the same scene. Discovering these associations over streams of video raises three new issues, which extend traditional techniques.

1. Common behaviors describe *activities* - The similar actions of different moving objects discovered in the stream (e.g., a car moving from right to left) need to be categorized under a common behavior called an *activity*.

2. No *a priori* knowledge of activities - The activities are not known in advance, rather they depend on the moving objects present in each video. Some method or model is needed to automatically infer activities from a video.

3. Users expect real-time processing and query answering - Answers to user queries about activities must be timely. Since there is a vast amount of data in a video stream, and the data arrives continuously, a method is needed to incrementally discover activities in a single pass over the data stream. Users will be satisfied to have an approximate answer with guaranteed low error value.

These challenging issues motivate our design of a framework for the real-time analysis of streaming video. Our visual surveillance system is designed to automatically answer questions such as: "Which is the most frequent scene seen so far?", "How important is that scene in the stream?", and "Is there a rule that explains the interactions between activities that are also seen in similar scenes?" To do this every scene is modeled as a combination of zero or more activities made by individual moving objects. That set of activities circumscribes the interactions between activities found in an scene and its frequency represents its importance over the stream.

In this chapter, we tackle the problem of finding the rules that govern the co-occurrence of activities in a continuous video stream with no prior knowledge of the number of activities. As we shall show, the discovery of activities is an off-line process in which event distributions are grouped. The discovery of rules is a on-line process that approximates the importance of each rule with guaranteed error value.

5.1.1 Contributions

This chapter makes the following contributions.

- We propose an unsupervised framework that efficiently addresses the complete process of scene understanding over video streams. Previous research (see Section 5.5) proposes either time consuming algorithms that are hard to scale to a stream or assumes a fixed number of activities in a video.

- We propose an algorithm to extract rules in a single pass over a video stream approximating the frequency of the activities found in a video with lower/upper bounds and providing guaranteed error values.

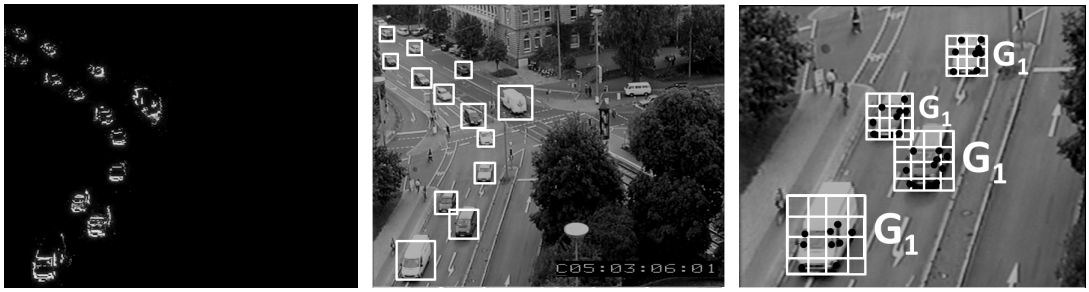
- We provide evidence that shows that hierarchical Bayesian models, including the hierarchical Dirichlet Process (HDP), are non-parametric Statistical models that suit the problem of activity recognition and visual surveillance. Most previous research has considered the generation of topic models under those models for large text datasets. We extend those findings to a noisier domain such as video databases and demonstrate that a hierarchy of two processes is needed to automate the discovery of activities.

5.1.2 Paper organization

The rest of this chapter is organized as follows. Section 5.2 describes a method to discover activities from video and provides an intuitive example to explain the need of a hierarchy for this problem. Section 5.3 discusses the frequency analysis of activities incrementally discovered from streams of video. Section 6.6 experimentally demonstrates the usefulness of the proposed approach for real traffic video. Section 5.5 discusses related work. Section 5.6 indicates limitations and possible extentions to this work. Finally, Section 6.7 concludes the paper.

5.2 Discovery of Activities

The problem of discovering activities in a video involves three kinds of information: *events*, *actions*, and *activities*. An *event* is an low-level interest point that represents a pixel with high variance in its spatio-temporal neighborhood. For a moving object, events occur in a bounding box forming a particular spatial arrangement of points that characterizes the *action* being performed. While a set of events characterizes an *action* (e.g., a car moving from right to left or a person walking in certain direction), *activities* are clusters of actions with similar event representation. This terminology and hierarchical relationship between *events*, *actions*, and *activities* have been adopted by the Computer Vision community, so we too use these common definitions. Given an input video, we take two consecutive frames and use a threshold to remove pixels with low intensity, as shown in Figure 6.1 (a). Then, we extract their events (gradient points) using a technique by Laptev et al. [46]. We evaluate connected components in Figure 6.1 (a) (represented as bounding boxes) to find moving



(a) Segmentation (b) Bounding boxes (c) Motion grids and Events

Figure 5.1. Describing activities of moving objects with events. (a) Motion information is segmented with pairs of consecutive frames (b) Bounding boxes enclose activity information (c) When zooming-in the frame, we can see how the spatial arrangement of events describes similar activities within 4×4 bounding boxes.

objects in the scene as shown in Figure 6.1 (b). Finally, we place grids on those boxes to discretize the location of existing events into $n \times n$ small regions, as shown in Figure 6.1 (c). Note that we want that every connected component often corresponds to a single moving object, so we obtained better results by only considering rectangular bounding boxes, enclosing components, with width-to-length ratio in the interval of $(0.7, 1.3)$. When we divide the number of events found in every small region by the total number of events in a motion grid, we estimate the probability of finding an event in that region. For objects performing the same activity (G_1) in Figure 6.1 (c), we can see how the grids also show a similar spatial arrangement of events.

Our goal at this stage is to model how events are organized into activities. Thus, in this section we use a hierarchical model of two levels to generate activities in video as multimodal probability distributions over events. The following example intuitively explains how this hierarchy works.

Example 2. *Let G_0 be a collection of possible traffic activities $G_0 = \{$TurnRight, TurnLeft, GoSouth$\}$ present in a video. We model a video as a sequence of combinations of activities with temporal order. Thus, instances of activities in G_0 are sparse over video frames forming the next sequence of actions: $G = \{[TurnLeft_{11}, TurnLeft_{12}], [TurnRight_{21}, TurnLeft_{13}, GoSouth_{31}], \cdots\}$, where [.] delimits a scene as shown in Figure 5.2. Each action*

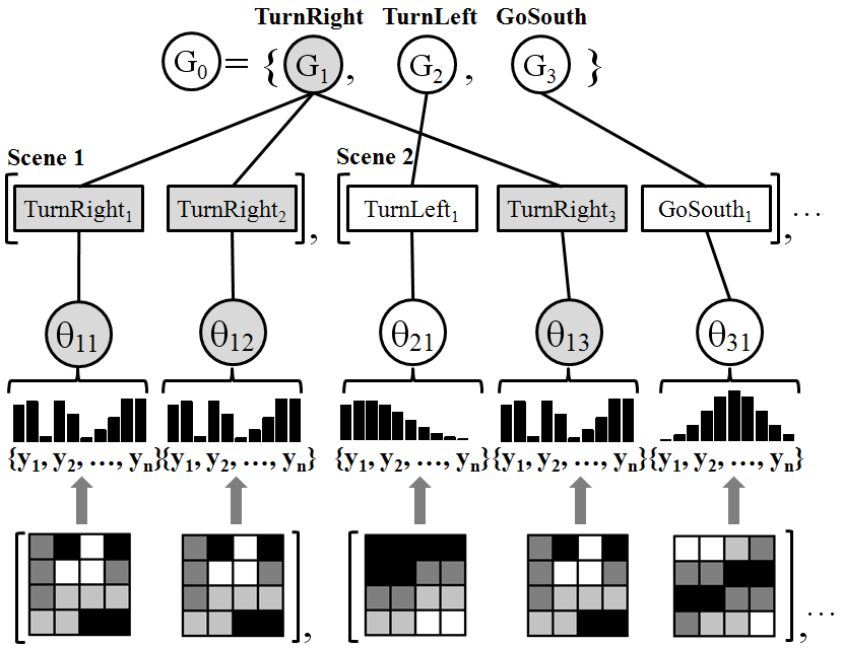


Figure 5.2. An example that shows the hierarchical process to discover activities in video.

in the scene is described by an $n \times n$ bounding box, whose row-wise traversal defines a histogram $y = \{y_1, y_2, ..., y_{n \times n}\}$ that represents the multimodal probability distribution θ_{ji} describing the activity j in terms of the $i-th$ position of a motion grid. In this example, the turn-left activity G_1 is formed by three similar (but not identical) turn-left actions $\{TurnLeft_1, TurnLeft_2, TurnLeft_3\}$, which have similar event distributions (represented as histograms in the lowest level of the hierarchy). Hence, the locations of events within a bounding box effectively characterizes those actions as instances of the activity G_1. Activities G_2 and G_3 are formed in a similar way.

The above example gives us an intuition about the hierarchical dependency between events, actions, and activities involved in this problem and models it with two levels in Figure 5.2. The lower part of the hierarchy generates a mixture of events y_i that uniquely defines an action with multimodal distribution θ_{ji}. The second level generates a list of activities G_0 distributed as the mixture model G_j over several multimodal distributions θ_{ji}. These two groups of information come from different, but related mixture models.

The hierarchical way of forming activities seems to indicate that both groups share some mixture parameters. However, note that we do not know the number of mixture component in G_0 needed to represent the clustering process involved. In our case, it is difficult to specify *a priori* the number of event observations (regions in a grid) and activities needed to correctly interpret interactions in a traffic video. Our approach is to set the number of event observations as an external parameter dependent on the resolution of a particular video, but infer the number of activities by using a Dirichlet process in each group of actions. The use of a Dirichlet Process is justified by its property of providing a non-parametric estimation of the number of mixture components for groups of observations.

We first define the Dirichlet Process and then present a hierarchy of two Dirichlet Processes that can discover a number of activities in video.

Dirichlet Process

Each event observation can be generated independently by a mixture component θ_{ji}. Let θ be a mixture component (cluster) associated to the event observation y_{ji}

Definition 4 (Dirichlet Process)**.** *A Dirichlet Process (DP) is a stochastic process that generates a distribution G in the form of an infinite mixture of components $\theta_i = \{\theta_1, \theta_2, ...\}$, a base distribution G_0, and a positive scaling parameter α.*

The construction of the Dirichlet Process can be formulated with sequences of independent random variables $(\pi'_i)_{i=1}^{\infty}$ and $(\theta_i)_{i=1}^{\infty}$, as originally stated in [47]:

$$\pi'_i \mid \alpha, G_0 \sim Beta(1, \alpha)$$

$$\theta_i \mid \alpha, G_0 \sim G_0$$

such that the random distribution G is then defined as:

$$\pi_i = \pi'_i \prod_{l=1}^{i-1} (1 - \pi'_l)$$

$$G = \sum_{i=1}^{\infty} \pi_i \delta_{\theta_k}$$

where δ_{θ_i} is an atomic distribution centered on θ_i. For convenience, we shall abbreviate the construction of π as $\pi \sim GEM(\alpha)$. Note that θ_i is a multinomial probability distribution over event observations y_i. In other words, the random variable θ_i has a probability of being associated to the set of event $y = \{y_1, y_2, ..., y_{n\times n}\}$. Hence, the distribution base G_0 also needs to be distributed as a multinomial distribution. This property of having a family of multivariate probability distributions is especially found in the Dirichlet distribution[1]*, so we model G_0 as being distributed as that distribution, $G_0 \sim Dirichlet(D_0)$.*

The Dirichlet Process generates a list of clusters of events $\theta = \{\theta_1, \theta_2, ...\}$ from the mixture model G that characterizes an activity based on the event observations y_i. Although this setting can represent appropriately one activity, it cannot represent several activities, which is needed for activity recognition in video. The modeling of activities is defined as a hierarchy of two DPs that relates the generation and activities jointly.

5.2.1 The Hierarchical Model

We employ the Hierarchical Dirichlet Process (HDP) introduced by Teh et al. [48] to mutually learn both actions and activities by considering a second DP which models groups of actions θ_{ji} into activities G_j. The result is a hierarchical process which can be understood as the two level DP represented in Figure 5.3.

The lower level of the hierarchy generates an unbounded number of HMMs (Hidden Markov Models) that learn activities with an unknown number of states, considering event probabilities from a motion grid as observable variables. The upper level combines similar actions (learned in the HMM) into activities.

Lower Level. The first level in the hierarchy constructs a variant of the Hidden Markov Model with state transitions distributed as G_j. The HMM is a doubly stochastic Markov chain in which a sequence of state variables $x = \{x_1, x_2, \ldots, x_T\}$ is hidden, but the

[1] This is the reason why a Dirichlet distribution is commonly denominated as a distribution of distributions.

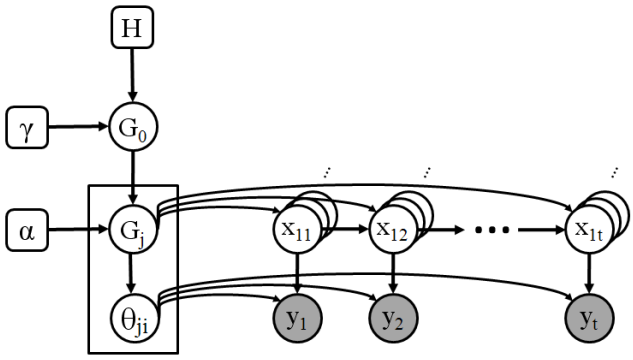


Figure 5.3. A hierarchical process to find activities in video. Each circle is a random variable and shading represents events observations from a grid.

sequence of observations $y = \{y_1, y_2, \ldots, y_T\}$ is observable. Changes between states are modeled with state transition probabilities and every state x_i is a multimodal variable that emits a discrete set of observations with some probability distribution. Traditionally, HMM assumes a Gaussian distribution for this property. However, it could represent even more complex observation behaviors when the output of the states is represented as the mixture of two or more Gaussians.

Every HMM is defined by the probability of each state to transit to other states and the probability of each state to emit an observation. In our model, both groups of information are assumed to be distributed as probability mixture models G_j for states and θ_{ji} for observations. A Dirichlet Process is used to approximate each mixture model with an unknown number of mixture components. Since we do not assume an arbitrary number of states, the transition to an infinite number of states is modeled using a DP following the construction procedure presented in Definition 4.

$$G_0 \mid \gamma, H \sim GEM(\gamma)$$

$$G_j \mid \alpha, G_0 \sim DP(\alpha, G_0)$$

$$\theta_{ji} \mid \alpha, G \sim DP(\alpha_0, G_j)$$

for each $j = 1, 2, ...$, the probability θ_{ji} related to the activity j are learned with a HMM of states x and observations y, which have the following distributions.

$$x_t \mid x_{t-1}, (G_{ji})_{j=1}^{\infty} \sim G_{x_{t-1}} \qquad \text{for states}$$
$$y_t \mid x_t, (\theta_{ji})_{j=1}^{\infty} \sim F(\theta_{x_t}) \qquad \text{for observations}$$

here, G_j is the distribution for the squared matrix that represents the transitions between states for the activity j. Different activities will be learned by HMMs with different distributions G_j.

Upper Level. While the lower level generates a list of HMMs that recognizes individual activities, the upper level in the hierarchy selects the optimal HMMs associated to the activity j. The result is a list of activities $G = \{G_1, G_2, ...\}$ distributed as a mixture model G_0 with base distribution H, and a positive scaling parameter γ.

$$G_0 \mid \gamma, H \sim GEM(\gamma)$$

In other words, the base distribution G_0 generates the distributions G_j by grouping similar HMMs that learns similar event distributions θ_{ji}. Teh et al. [48] also use Gibbs sampling schemes to do inference under the HDP model. To detect the activity associated to a bounding box with a sequence of events observations $\{y_1, y_2, ..., y_{n \times n}\}$, first the trained HMM with highest log-likelihood score is selected. Second, the activity of the corresponding G_j associated to the item j is chosen.

5.3 Discovery of Interactions

The output of the hierarchical model present in Section 5.2 is able to discover the activities that positively correlate in frequent scenes. The discovery of important interactions between those activities can take at any time over the video stream as scenes continuously arrive. Thus, analysis of this data must be incremental as multiple scans to learn the underlying interaction model of the video stream would be too expensive. The algorithm proposed in this section extends the well-known Apriori algorithm [49] by generating ap-

proximate rules over a video stream in a single pass and with guaranteed error value. The output of the algorithm is a subset of dynamic activity interactions with an importance value over the video stream at time t.

The algorithm is based on the plausible idea that a video stream is formed by a sequence of short scenes in which activities are simultaneous. Two or more consecutive frames define a *scene* that contains subsets of the set of activities learned in Section 5.2 with a hierarchy of two Dirichlet Processes. Those subsets of activities are denominated *activity sets* in this chapter to represent a spatio-temporal environment for co-occurring activities. Note that complex activities do not occur solely in one scene, but they propagate their behavior over consecutive ones. Hence, we define a *transaction* as a larger time window that contains a sequence of scenes. This is a range that specifies the maximum allowed time difference between the earliest and latest occurrence of activities.

Two important properties of activity sets are their *support* (f) and *error value* (ϵ). While the *support* refers to the approximate number of transactions that contain an activity set, the *error* value is guarantees that the approximate results will not exceed that number. Frequent activity sets are those with support value above some threshold. We want to compute rules of the form $set_1 \rightarrow set_2$ such that $set_1 \cup set_2$ has high support.

To compute approximate support values in a single pass, we need to store frequent counts of activity sets previously seen in the stream into a data structure D. Note that D can be any data structure that efficiently implements the operations $exist()$, $get()$, $insert()$, and $update()$. Commonly, a prefix tree data structure is used to store activity sets in lexicographical order along with their frequency counters in $O(1)$ [50]. Entries in D have the following format:

$$< set, \overline{f}_{set}, \Delta_{set} >$$

where set is an activity set, $\overline{f}_{set}$ is a lower bound for the real frequency value f_{set}, and Δ_{set} represents the difference between its upper and the lower bounds. Note that those bounds define an interval that contains the real frequency value f_{set}.

$$\overline{f}_{set} \leq f_{set} \leq \overline{f}_{set} + \Delta_{set}$$

Later, we will show that the upper bound can be reduced to ϵN, where N is the number of transactions seen so far in the stream and ϵ is the desired error value.

The algorithm starts by logically dividing the incoming stream into transactions of $w = \frac{1}{\epsilon}$ scenes. The set of activities, let's call it *cand*, contained in an incoming scene represents recent information to be analyzed by its frequency count in the current transaction. To do that we first evaluate its power set $\mathcal{P}(cand)$, excluding the null set, to evaluate the possible combinations of activities that may be frequent. Note that the power set of a scene does not produce a huge number of activity sets C since a scene commonly contains only the interaction of a limited number of activities. The index of the current transaction being analyzed is denoted as $b_{current}$ and its value is computed as $\lfloor \frac{N}{w} \rfloor$.

For every activity set *set* in C, the algorithm looks up *set* in D to know whether it exists or not. When *set* is seen for the first time, and therefore it is not in D, we check whether *set* has a high support value in the w scenes of the current transaction. If so, we assign the lower bound on its frequency as $\overline{f}_{set} = 1$ and the extension of its confidence interval as $\Delta_{set} = b_{current} = \lfloor \frac{N}{w} \rfloor$. This indicates that we have currently seen *set* at least once on the stream and at most once on every transaction, respectively. The lower bound $\overline{f}_{set}$ will increase when new instances of *set* are found in later windows.

If *set* is currently stored in D, its lower bound $\overline{f}_{set}$ is increased by 1. The extension of its interval is updated by the difference between its current and former transaction indices such that a low uncertainty value is obtained when a set recently occurs often and a high uncertainty value is computed when there is a large gap between occurrences. An upper bound below $\lfloor \frac{N}{w} \rfloor$ (i.e., the current number of time windows processed so far) is an indicator that the activity set may have been frequent in early transactions, but has not been updated recently. Hence, we delete those entries with $\overline{f}_{set} + \Delta_{set} \leq \lfloor \frac{N}{w} \rfloor$ to only generate co-occurring rules based on recent and frequent activity sets.

Every activity set *set* is inserted in a priority queue Q of size K. Subsequently, after

all the scenes in the current transaction have been processed, Q will automatically retain the K most frequent activities sorted by their approximate support $\overline{f}_{set}$. The process of estimating approximate activities is summarized in Algorithm 3.

The use of lower and upper bounds assure us that after reading N scenes in the stream, those activity sets whose frequency exceed $\lfloor \frac{N}{w} \rfloor$ will be stored. Since $w = \frac{1}{\epsilon}$, the threshold is $\lfloor \frac{N}{1/\epsilon} \rfloor$ and thus $\lfloor \epsilon N \rfloor$ is a guaranteed worst-case approximation on the value of f_{set}.

Algorithm 3 SetCounting(T, D, N, w)

```
1: //T: current transaction
2: for i = 0 to w − 1 do
3:    cand ← T_i; //activity set cand at the i^th scene
4:    b_current ← ⌊N/w⌋;
5:    //power set on each scene
6:    C ← P(cand) − {∅};
7:    for j = 0 to /C/ do
8:       set ← C_j;
9:       f̄_set ← count(set);
10:      //Inserting activity set
11:      if !D.exist(set) & f̄_set ≥ ⌊N/w⌋ then
12:         D.insert(< set, f̄_set, b_current >);
13:      else
14:         //Updating approximate support
15:         old ← D.get(set);
16:         f̄_set ← f̄_old + f̄_set;
17:         interval ← b_current − (f̄_old + Δ_old);
18:         D.update(< set, f̄_set, interval >);
19:      end if
20:      //Pruning condition
21:      if f̄_set + (b_current)_temp ≤ (b_current)_set then
22:         D.remove(set);
23:      end if
24:      if D.get(set) ≠ {∅} then
25:         Q.push(< set, f̄_set, Δ_set >);
26:      end if
27:      N ← N + 1;
28:   end for
29: end for
30: return Q;
```

Given the frequent activity set $set = set_1 \cup set_2$ seen so far in the stream, we want to represent its interactions in the same scene by $set_1 \rightarrow set_2$ ($set_1 \cap set_2 = \emptyset, set_2 = set - set_1$).

The importance of this abstraction is measured by its *confidence*.

$$confidence(set_1 \rightarrow set_2) = \frac{f_{set1 \cup set2}}{f_{set1}}$$

Note that this value is not exact since its evaluation relies on approximate frequency counts, so we consider a guaranteed worst-case approximation of $\overline{f}_{set} \pm \frac{\lfloor \epsilon N \rfloor}{2}$. The process of incrementally producing rules for a current transaction is summarized in Algorithm 4. In our experiments we use the *confidence* and *support* of a rule to determine its importance.

Algorithm 4 ApproximateRuleExtraction(Stream S)

```
1: D ← 0; //D: data structure
2: Q ← 0; //Q: priority queue
3: N ← 0; //N: number of scenes so far
4: w ← 1/ε; //w: number of scenes in T
5: while true do
6:    Q ← SetCounting(S.getTransaction(), D, N, w);
7:    while Q.hasElements() do
8:       < set, f̄_set, Δ_set > ← Q.pop();
9:       for every nonempty subset set_1 of set do
10:         set_2 = set − set_1;
11:         if confidence(set_1, set_2) ≥ confidence_min then
12:            print(rule : set_1 → set_2);
13:            print(support : f̄_set ± ⌊εN⌋/2);
14:         end if
15:      end for
16:   end while
17:   N ← N + w;
18: end while
```

Note that while the Apriori algorithm uses the anti-monotone property to avoid computing all possible combinations of activities, we evaluate all possible subsets of activities in a scene in seek of approximate their counting over the video stream. However, we reduce the candidate space by keeping only those that are frequent and recent over the stream. We thus assume that if a subset of activities is frequent in a transaction, it could also be frequent during the incoming part of the stream. Those subsets that are frequent, but become outdated, are removed as candidates. The inserting of frequent candidate subsets in Q provides frequent interactions for the current transaction, where we can discover rules

with high confidence values as the stream arrives.

Note that $set_1 \rightarrow set_2$ defines the co-occurring interactions between the antecedent and consequent part of that rule and not the estimate of the probability of finding set_2 under the condition that these transactions also contain set_1. This information could also be modeled as an additional layer in the Topic Model described in Section 5.2. However, the resulting model would not be as dynamic and incremental as the one proposed in this chapter. Our algorithm, rather than solving the efficient storing of frequent itemsets, aims to make its working adaptable to the limited resources of a system while providing bounds and a guaranteed error for frequency values. We study this problem in detail in the next subsection.

5.3.1 Dynamic Memory Usage

The approximate counting of frequent itemsets helps us to understand the significance of scenes in a video. However, both the detection of items in the video (Section 5.2) and their posterior frequency evaluation (Section 5.3) have different memory requirements. While the detection of itemsets takes consecutive frames to recognize items based on events, the approximate counting of itemsets is much faster and provides better approximations when more time windows are considered. To illustrate this concept consider Figure 5.4 as an example. A stream of video frames arrives continuously as input. We trained the Dirichlet Process introduced in Section 5.2 to recognize itemsets by assigning labels to activities. The output of this process is a time window of w transactions.

We can increase the probability of finding itemsets in several transactions by reading $B > 1$ time windows. This also provides a more effective pruning of those itemsets with supports below than $(s-\epsilon)N$. Thus, we store in a queue as many time windows as possible, being $B = 1$ the lowest value. However, memory space is a limited resource that may hinder the evaluation of approximate frequency counts when few transactions are considered. This problem is especially important in our case because of the amount of memory already devoted to recognize activities in a continuous stream of video frames. This makes dynamic the amount of memory available in the system with values even less than required to store

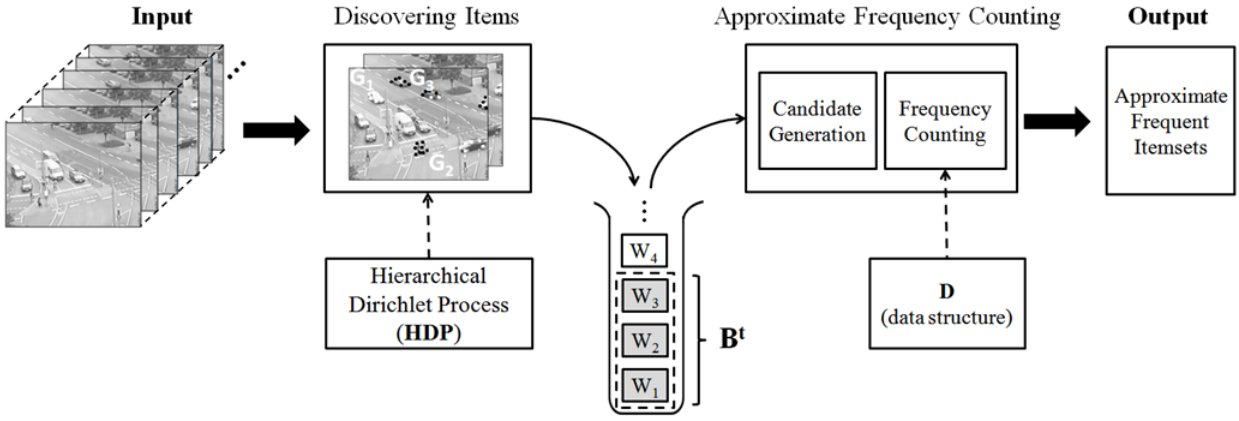


Figure 5.4. A broad representation of the approach introduced in this chapter. A continuous video stream is discretized in windows B that contain a variable number of transactions at time t to approximate the frequency counts of itemsets in the window.

one time window.

Thus, there is a trade-off between storing as many windows as possible, to better approximate frequency values, and compute the frequency of itemsets within a window as soon as one time window is received. An intermediate solution is to dynamically adjust the parameter B to control the number of time windows to be stored in the queue.

If the number of windows is less than B and enough memory is available, then we keep inserting time windows in the queue. If the amount of memory is almost full and the number of windows in the queue is less than B, then we delete those windows to avoid storing older frequencies. If memory is not full and there are more than B windows in the queue, we keep inserting windows until some threshold is exceeded and then take the total number of windows in the queue to improve the approximation on frequent counts.

Let M_B be the space in memory allocated to store B windows, $M_B = wB$. A windows w contains $\frac{1}{\epsilon}$ transactions with at most $/I/$ items (i.e., possible types of activities). Thus, the maximum amount of memory to allocate a window is $w = \frac{/I/\cdot 1}{\epsilon}$ transactions and therefore we store $M_B = \frac{B/I/}{\epsilon}$ units in the queue. The amount of available memory at time t is then defined as the difference between the total memory at time t and the memory employed to

store B time windows at time t.

$$M^t_{available} = M^t_{total} - \frac{B^t/I/}{\epsilon}$$

Assuming that the last state in the queue, in which we took B time windows, is stable, we want to preserve the balance between windows stored in the queue and the corresponding amount of memory available:

$$\frac{\#transactions^{t-1}}{M^{t-1}_{available}} = \frac{\#transactions^t}{M^t_{available}}$$

once $M_{available}$ and B are considered in the above equation, we obtain the amount of time windows needed to read in memory to assure the stable working of the system at time t.

$$B^t = \frac{M^t_{total} B^{t-1}}{M^{t-1}_{total}}$$

Note that although both $/I/$ and ϵ are fixed parameters, we can consider B^t as a running parameter in our model. An inadequate control of B may hinder the performance of the system since it affects the number of transactions that need to be created, updated, or removed from the data structure D. This is especially important when D is implemented on hard disk, a common scenario when very large stream databases are considered.

By modeling the allocation of transactions in memory in terms of B, we make this process adaptable to that limited resource. We would like to always set B to a high value to have more accurate approximations for the counts of itemsets, but this value needs to be decided according to the total amount of memory in the system at time $t-1$, which is not constant.

5.3.2 Summary

In this section, we have computed frequent itemsets with approximate support and guaranteed error value while adjusting dynamically the number of transactions needed to effectively evaluate support values. Intuitively, the more transactions we take, the more

precise frequency approximations we obtain. However, more available memory will be consumed as well. Our method reduces space complexity by adapting the algorithm to the amount of memory available before any process to update frequency values for itemsets is incrementally performed.

5.4 Experiments

In this section, we test the performance of our technique with outdoor videos where moving objects describe traffic scenes governed by the state of multiple semaphores. The co-occurring interactions are modeled by frequent sets of activities with large confidence values over the video stream. We experiment on the following datasets: ***Street Intersection***[2] (*normal quality, 25fps, 45 minutes, 5 semaphores*), ***Karl-Wilhelm*** & ***Strabe Streets***[3] (*normal definition, 25fps, 1 hour, 3 semaphores*), and ***Roundabout Junction***[4] (*normal quality, 25fps, 1 hour, 3 semaphores*).

The experiments are run on a 3.6 GHz Pentium 4 with 2 GB RAM and all the above datasets are publicly available to facilitate later experimental comparisons.

5.4.1 Experiment 1: Discovering activities

Before studying the quality of the co-occurring relationships generated to describe scenes in videos, we analyze in this experiment the recognition of activities with the model presented in Section 5.2. We compare our instantiation of the hierarchical Dirichlet Process (HDP) to another probabilistic topic model, the Latent Dirichlet Allocation (LDA) [51], recently used in [52–54] to also segment visual features from video into activities, and a supervised approach, the Support Vector Machine (SVM). Most existing research employs hierarchical Bayesian models or non-probabilistic classifiers to recognize activities in video, so by using both LDA and SVM we cover two representative techniques in the literature.

We used the first 20 minutes of each dataset for training in order to take the remaining data for testing by labeling and recognizing activities on it. Pairs of consecutive frames are

[2] http://www.eecs.qmul.ac.uk/~jianli/Junction.html
[3] http://i21www.ira.uka.de/image_sequences/
[4] http://www.eecs.qmul.ac.uk/~jianli/Roundabout.html

Table 5.1. Information on the Training Stage for Each Dataset.

Dataset	Activities	Time to Compute
Street Intersection	37	4.38 min.
Karl-Wilhelm & Strabe	31	3.51 min.
Roundabout Junction	24	2.57 min.

processed to identify moving pixels, events, and connected components. The observations consist of bounding boxes around moving objects with resolutions of 8×8 for the **Karl-Wilhelm** & **Strabe** dataset and 4×4 for the **Street Intersection** and **Roundabout Junction** datasets. This is because the camera in the first dataset is placed on a far building, so we need grids with higher resolutions to describe small objects. This process provides a collection of unlabeled motion grids to the hierarchical model. We do not assume any prior knowledge in the number activities to be discovered. DP parameters were fixed at $\{\alpha = 11, \gamma = 0.9\}$.

Table 6.1 shows the number of activities found and the time consumed by each dataset. Since LDA and SVM cannot find the number of activities from the video, we first use HDP to discover the number of activities and then employ K-means to generate a number of clusters in the training dataset and assign them a label. Finally, we train SVM with these already labeled groups and perform generalization in the remaining video to assign labels to moving objects, as also described by Yin and Meng [53]. By contrast, LDA only needs the number of activities provided by HDP to be trained and recognize activities. The finding of activities by these three techniques is measured by comparing their clustering "goodness" on the testing data considering the same number of activities and increasing database size.

Figure 5.5 shows the result of grouping activities in each dataset with the trained HDP, LDA, and SVM. Since there is not a direct way to compare the accuracy of two clustering algorithms, we estimate the *average clustering error* by averaging the distance between the activities that belong to each cluster by their total number of cluster elements. This number indicates how similar are the actions, represented by their event distributions, which belong to the same cluster. A large value will indicate poor clustering since the actions within a

cluster will be heterogeneous indicating large entropy in the cluster. Hence, an algorithm with a low *average clustering error* is preferred. We computed the error along the duration of each video and plotted the result in Figure 5.5. The technique used in this chapter provides the lowest error value for every dataset, but this difference is smaller for datasets with fewer activities. For instance, while we can clearly see the advantage of using HDP to detect activities in the **Street Intersection** dataset (37 activities) and the **Karl-Wilhelm & Strabe dataset** (31 activities) in comparison to other methods, the discovery of activities in datasets with fewer activities, like the **Roundabout Junction** dataset (24 activities), produces comparable error values in each method. This behavior was also noticed by Wang and Mori [55] when comparing different Bayesian models for human activities recognition in video. Additionally, LDA seems to have a better behavior than SVM with K-means in all the datasets considered in this chapter.

5.4.2 Experiment 2: Discovering Interactions

In this experiment, we study the significance of the generated rules to understand the dependencies between the activities discovered in Experiment 1. Transactions of size $/w/ = 25$ scenes is a value that works in all the datasets in order to find activities temporally correlated in the same window. Thus, every transaction is the input to the Algorithm 3 and since $w = \frac{1}{\epsilon}$, we approximate the frequency counts of every activity set with an error value of $\epsilon = 0.04$.

The number of transactions, topics, and the processing time to discover association rules for every dataset are summarized in Table 6.1. The **Street intersection** dataset exhibits more topics than the **Karl-Wilhelm & Strabe** dataset since five traffic lights decomposes complex activities into a large number of well-defined scenes. On the other hand, the **Roundabout Junction** dataset contains a few number of topics due to the limited types of activities performed and considerable amount of frames with no activities. The processing time to generate rules seems to be proportional to the number of topics discovered in each dataset.

We consider a minimum support value of 4%, a minimum confidence value of 90%, and

Table 5.2. Information on the Datasets Preprocessed to Discover Association Rules.

Dataset	**Time windows**	**Topics**	**Rule generation time**
Street Intersection	$\sim$ 15000	37	8.74 sec.
Karl-Wilhelm & Strabe	$\sim$ 11000	31	5.41 sec.
Roundabout Junction	$\sim$ 7000	24	3.28 sec.

activity clusters with more than 10 elements in order to generate representative rules. We thus extract rules from the **Street Intersection** (37 topics and 16 rules), **Karl-Wilhelm & Strabe** (31 topics and 13 rules), and **Roundabout Junction** (24 topics and 10 rules) datasets. In these datasets, as more constraints govern the activities (e.g., traffic lights, one-way roads, intersections, etc.), more topics will be generated and more frequent rules will be discovered. This evidence seems to indicate that every constraint imposes an underlying logic that fragments complex activities into a large number of small scenes, which are easy to represent with events and form well-defined activities, and therefore are likely to be frequent during the video. For the **Street Intersection** dataset, some of the rules uncovered with the algorithm proposed in this chapter are depicted in Figure 5.6 (a) and detailed in Figure 5.6 (b). The first three rules are high-confidence associations that suggest a strong correlation between vehicles moving in parallel lanes (G_2 and G_3) or those moving from side to side (G_7 or G_{11}) while other vehicles move away from the center to the top left of the scene (G_3). Those activities are mutually exclusive since there are five traffic lights that prevent vehicles moving from side to side from colliding with those moving across the parallel lanes. Consider the first rule $\{G_7\} \rightarrow \{G_3\}$ as an example. The last two rules indicate the co-occurring dependency between cars turning right (G_5) while others that are moving from right to left (G_7). This behavior, exemplified by rule $\{G_5\} \rightarrow \{G_3, G_7\}$, is justified since those vehicles use the same traffic light to move from the bottom right part of the scene to either the bottom left or the top left edge, as seen in scene 4 and 5 of Figure 5.6.

For the **Karl-Wilhelm & Strabe** dataset, three confident interactions are shown in Figure 5.7 (a) and expressed with rules in Figure 5.7 (b). We notice the regular presence of the activity G_5 in those scenes. This behavior is reasonable since the activity G_5 corresponds

to vehicles going along Strabe avenue, a very busy road in the dataset. The first rule $\{G_2\} \rightarrow \{G_5\}$ exemplifies the interaction of vehicles going in parallel lanes without restrictions. The second rule is similar, but additionally contains the activity of cars going from the center to the bottom left of the screen (G_7). Furthermore, the usual interaction of cars going straight in the avenue (G_5) and then turning right after that (G_7) is explained by rule $\{G_5\} \rightarrow \{G_7\}$.

For the **Roundabout Junction** dataset, we show common interactions in Figure 5.8 (a) and detailed in Figure 5.8 (b). The roundabout in the video segments the motion of vehicles into multiple activities. The first rules $\{G_{11}\} \rightarrow \{G_3\}$ represents the sequence of car activities going straight (G_3) and then joining the roundabout (G_3). By contrast, the second rule $\{G_9\} \rightarrow \{G_2\}$ explains the co-occurring relationship of vehicles taking lanes separated by the roundabout. Finally, while some vehicles circulate alongside the roundabout emerging as two activities (G_5 and G_{14}), another set of cars take a different way by turning left from the center to the upper part of the scene (G_8). These sequences of co-occurring associations reflect the transitions between significant scenes in datasets governed by multiple lanes, traffic lights, and a roundabout.

5.4.3 Discussion

The time complexity for learning activities with the hierarchical Bayesian method used in Section 5.2 is: multiple

$$O(W) + O(EG) + O(G^2)$$

where W is the number of scenes used for training, E is the total number of regions in a motion grid, and G is the number of activities discovered. Note that the resolution of a grid will affect the number of event observations during training. The space complexity spent for this process is

$$O(GE) + O(G) + O(GW) + O(EW)$$

The time complexity to discover interactions in a transaction is:

$$O(2^G(W+1) + TKlog(K))$$

where W is the total number of scenes in a transaction and K is the size of a priority queue. We multiply that expression by the number of transactions seen so far to have a notion of the time complexity. We assume a constant insertion and searching time, as provided by a hash table, during the counting of activity sets. For every scene, G represents the maximum number of activities observed. Note that although the expression 2^G is the power set of activities and it may provide a large number of activity sets to be computed, in practice it is not common to find all the possible activities in the same scene. Thus, this number is small allowing us to obtain an approximation algorithm for this problem.

5.5 Related Work

In this section we compare our approach with related efforts. For clarity, we keep our comparison focused in each stage of the video process (i.e., discovery of activities and discovery of interactions). The recognition of activities in video is an open problem that has received much attention lately. Commonly, low-level visual features and actions have been modeled and classified to provide interpretation of activities. While the traditional way to categorize existing research is by motion representation such as local features (e.g., changes in velocity, changes in curvature of motion trajectories, and gradients) or global features (e.g., key frames), recent research has employed hierarchical Bayesian models such as LDA [51] and HDP [48] to cluster local motions into activities successfully, c.f., [52–54]. The above research has led to techniques that can discover atomic activities, but such techniques omit the complex interactions between activities commonly present in video. Wang et al. [54] approach this problem by adding one more level to the hierarchy of the LDA and HDP Bayesian models and providing extended versions of integral probabilistic hierarchical Bayesian models (LDA, HDP, and Dual-HDP mixture models) to cluster moving pixels into atomic activities and interactions. Similarly, Li et at. [56] infer global behavior patterns through modeling behavior correlations through a hierarchical probabilistic Latent Semantic Analysis (pLSA). Both techniques, however, learn global interactions disregarding temporal information. By contrast, our on-line technique relates frequent activities in a transaction and removes those that become infrequent over time. In other words, by decoupling both

the discovery of activities and interactions, we can incrementally learn interactions without assuming the same probability of co-occurring relationships over time, a reasonable scenario imposed by the processing of continuous video streams.

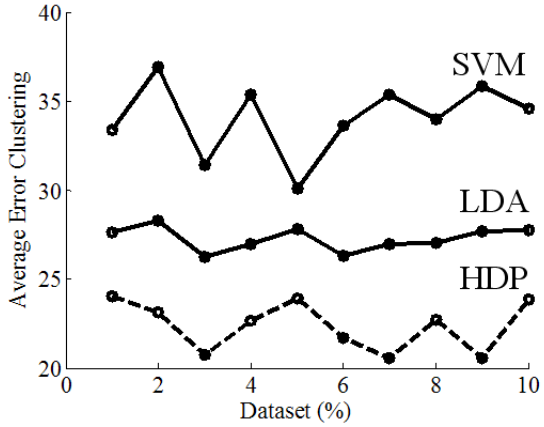
5.6 Limitations of this Work

To understand the behavior of moving objects in a video stream, we adopt events based on moving pixels as low-level visual features that characterize their spatial dependencies. While we have showed their expressiveness to discover activities and co-occurring relationships, others features can also be adopted to consider different attributes such as shape, trajectory, and appearance to distinguish between vehicles and people, detect when vehicles stop because of a red light, or when people describe two trajectories by crossing at a crosswalk in both directions. An extension of this work is to use trajectory and appearance information to differentiate activities (e.g. a car going left and a person going left). Additionally, the assumption of having a static camera recording activities is reasonable for traffic surveillance, but we will need to stabilize regions containing moving information, as in [55], to extend our work to more dynamic scenes. As future research, we plan to study other similar domains where crowds of moving objects also have different behaviors at the same time. Also, we want to study the relationship between frequent activities and recognition of abnormal situations, often related to dangerous events for the domain of traffic roads. Additionally, video summarization can also be implemented by providing to the user a sequence of important scenes in terms of confidence and support values.

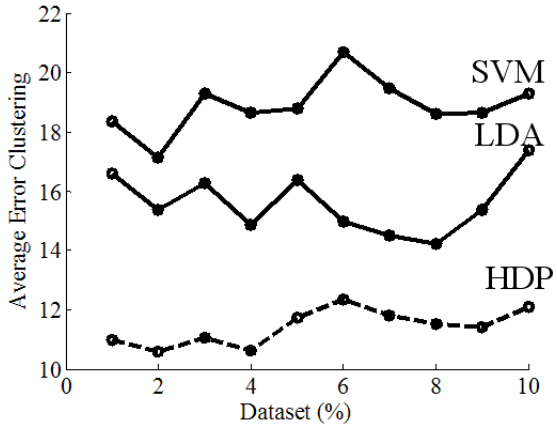
5.7 Conclusion

In this chapter, we propose a framework to find approximate co-occurring associations from video stream data considering unsupervised clustering of events (low-level visual features) into activities. We define activities as actions described by similar event distributions. A hierarchy of two stochastic processes is used to avoid considering an arbitrary number of activities in the video. The most visible aspect of this effort is the incremental generation of rules that discover the interaction of frequent activities for current scenes. Our experimental

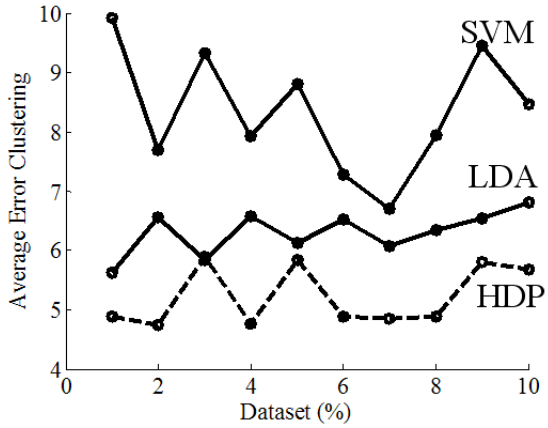
results show that our approach efficiently and automatically discovers sets of activities in a video stream while evaluating their frequent occurrence and co-occurring relationships with guaranteed error value.



(a) Street Intersection

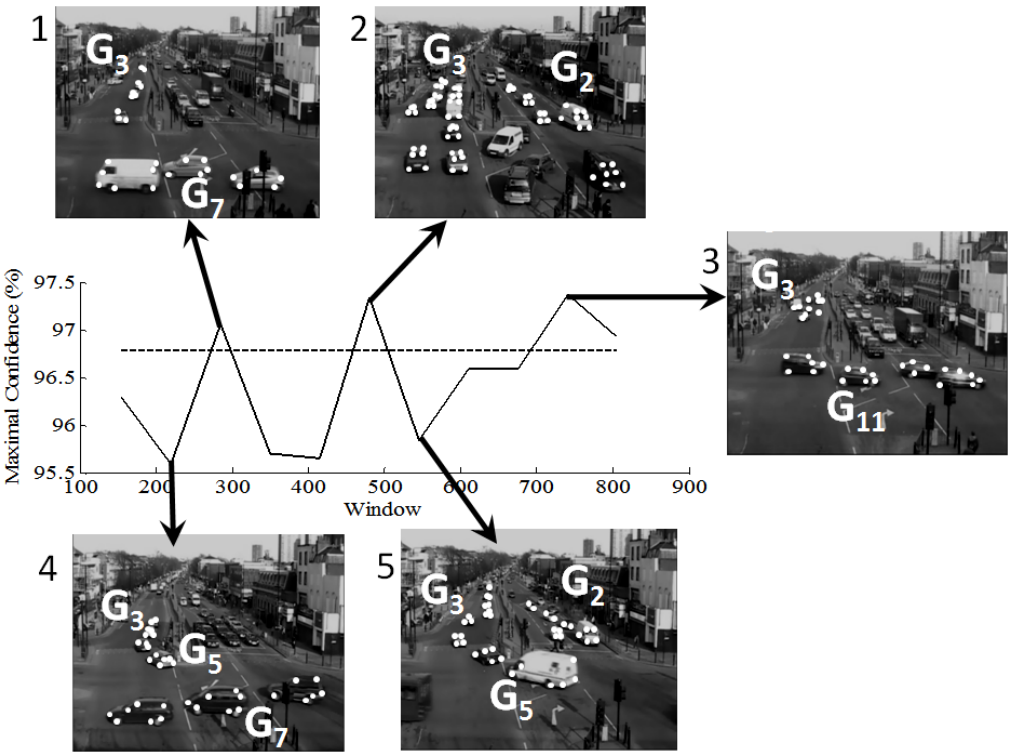


(b) Karl-Wilhelm & Strabe



(c) Roundabout Junction

Figure 5.5. A comparison of the average clustering error in each dataset. It measures the similarity of the elements within clusters.

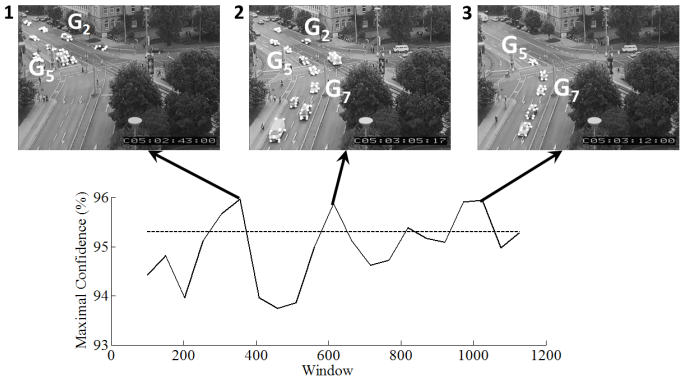


(a) Scenes

Scene	Rule	Confidence	Support
1	$\{G_7\} \rightarrow \{G_3\}$	97.2%	7.6%
2	$\{G_2\} \rightarrow \{G_3\}$	97.3%	11.2%
3	$\{G_{11}\} \rightarrow \{G_3\}$	97.4%	8.8%
4	$\{G_5\} \rightarrow \{G_3, G_7\}$	95.5%	6.5%
5	$\{G_3, G_5\} \rightarrow \{G_2\}$	95.6%	8.2%

(b) Rules

Figure 5.6. Experiment on the Street Intersection dataset. (a) A selection of high confidence association rules. (b) Scenes of the Street Intersection dataset with high confidence values over time. Changes between scenes represent transitions between significant scenes.

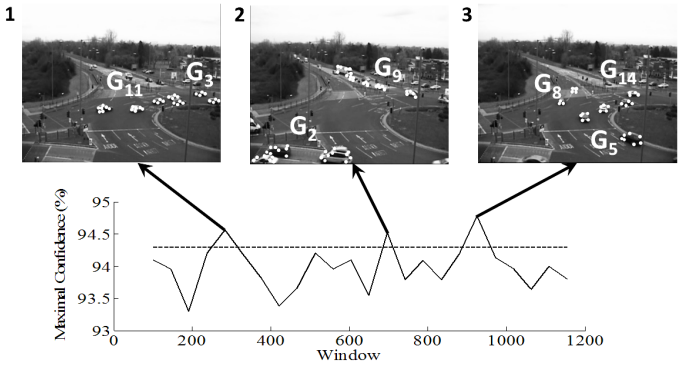


(a) Scenes

Scene	Rule	Confidence	Support
1	$\{G_2\} \rightarrow \{G_5\}$	95.9%	21.3%
2	$\{G_2, G_5\} \rightarrow \{G_7\}$	95.7%	17.8%
3	$\{G_5\} \rightarrow \{G_7\}$	95.6%	24.6%

(b) Rules

Figure 5.7. Experiment on the Karl-Wilhelm & Strabe dataset. (a) A selection of high confidence association rules. (b) Scenes of the Street Intersection dataset with high confidence values over time.



(a) Scenes

Scene	Rule	Confidence	Support
1	$\{G_{11}\} \rightarrow \{G_3\}$	94.5%	27.3%
2	$\{G_9\} \rightarrow \{G_2\}$	94.4%	13.5%
3	$\{G_5, G_{14}\} \rightarrow \{G_8\}$	94.8%	19.8%

(b) Rules

Figure 5.8. Experiment on the Roundabout Junction dataset. (a) A selection of high confidence association rules. (b) Scenes of the Street Intersection dataset with high confidence values over time.

CHAPTER 6
WHAT TO REUSE?: A PROBABILISTIC MODEL TO TRANSFER USER ANNOTATIONS IN A SURVEILLANCE VIDEO

6.1 Introduction

Surveillance videos are widely used in real-time monitoring applications, e.g., of an oil spill, a store entrance, or a traffic road. Due to the nature of surveillance, continuous recording is needed to capture events during days and weeks. This produces dynamic content that makes it difficult to reuse labeled data from other parts of the video to improve prediction in current scenes. The prediction of labels in surveillance videos raises three important issues which go beyond traditional machine learning algorithms:

1. Mixed membership. The mixture components that form a video scene are not known in advance; rather they should be learned based on the frequent co-occurrence of discrete activities in the video. This explains the need for a generative process, like Latent Dirichlet Allocation (LDA), to generate a proportion of topics for every scene.

2. Dynamic content. video is noisy and external situations produce different features (activities) persistently. As LDA provides topics based on feature co-occurrences, we obtain different topics at different places in the video. This makes it difficult to compare statistical models trained in different domains.

3. User annotations are scarce and expensive. We can ask users to provide labels for every scene via crowdsourcing, but usually those are limited to a small portion of the data. Repeating this process when every new external variable happens is not scalable

in a continuously growing video database. We rather want to improve precision by reusing labels of the existing domain that best fit our current data.

Based on the above observations, we propose the Crossdomain Probabilistic Model (CPM). The basic idea is to generate topics that model activities of two domains jointly. Those become the common latent variables to propagate labels from a source to a target domain. The similarity of two domains, in terms of the number of common activities, guides the inference of latent variables.

6.1.1 Contributions

This chapter makes the following contributions.

- Our probabilistic model is able to learn topics that explain the co-occurrence of activities across domains. Existing research performs prediction by considering a single domain [57, 58]. This leads to poor precision and recall values when marginal and conditional distributions change in the video.
- This model is able to represent the arbitrary relationships between domains with a covariance matrix that weights more the domains that share more observable variables in the content.
- We propose an approach that combines content and labels. Previous research (see Section 6.3) proposes algorithms that learn conditional dependencies between random variables to explain the content of video scenes, but omits the generation of labels by users. Both content and labels are observable variables in our problem that are learned jointly. This way, when users annotate a scene as dangerous, our model puts some bias in the corresponding topics that describe the scene.

6.1.2 Chapter Organization

The rest of this chapter is organized as follows. Section 6.2 describes important definitions used in this chapter to set a common terminology. Section 6.3 discusses related

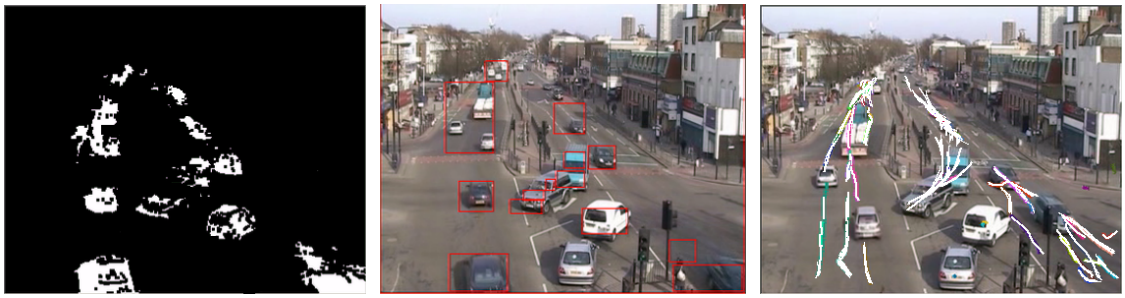
work. Section 6.4 analyzes three methods to connect content present in different domains. Section 6.5 introduces our probabilistic model. Section 6.6 shows experiments that demonstrate the usefulness of the proposed approach for real traffic videosets. Finally, Section 6.7 concludes the chapter.

6.2 Notation and Terminology

The problem of understanding video involves three kinds of information: *events*, *activities*, and *scenes*. An *event* is a trajectory that exhibits the temporal behavior of a moving object, for example, consider the trajectories of vehicles going up, down, and turning right in Figure 6.1 (c). Then, an *activity* groups trajectories with similar shape. To do that, we compute the hash key of every trajectory with the Timeseries Sensitive Hashing (TSH) algorithm proposed in [4]. TSH has the ability to map timeseries with similar shape, but different length (see Figure 6.2 (a)), to the same bucket of a hash table with high probability. As shown in Figure 6.2 (b), the result is a dictionary of discrete activities (e.g., A, B, C, and D) computed in linear time, where each entry in the vocabulary corresponds to a bucket in the hash table that contains a number of trajectories. Note that, the tuning of the parameters of TSH (e.g., bucket width and hash function) will create a dictionary with a given number of actives. Finally, a *scene* is a time window of fixed length that contains a set of activities. This terminology and hierarchical way of generating *events*, *activities*, and *scenes* has been adopted by the Computer Vision community, so we also consider these common concepts in this chapter. The reader can consider activities as the words and scenes as the documents of a text corpus.

Formally, we define the following terms to use in this chapter:

- An *activity* a_i is the basic unit of discrete data. It is an entry of a fixed dictionary of V terms.
- A *topic* β_k is a distribution over a subset of activities.



(a) Segmentation (b) Bounding boxes (c) Trajectories

Figure 6.1. Describing surveillance video for traffic roads. (a) Motion information is segmented with pairs of consecutive frames (b) Bounding boxes enclose activity information (c) Bounding boxes are tracked during a time window to extract trajectories, highlighted in white.

- A *scene* m is a time window containing a collection of N activities denoted by $A = \{a_1, a_2, ..., a_N\}$. We can generate a scene, m, as a proportion, θ_m, over K existing topics.

- An *annotation* is a numeric value $r = \{0, 1\}$ provided by a user to represent his/her perception of a scene (e.g., *is this scene dangerous?*).

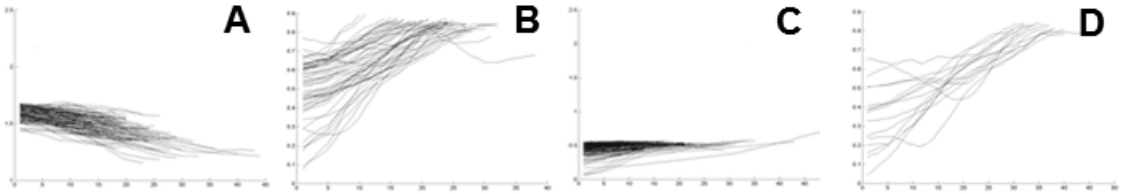
- A *domain* D is a matrix of users I as rows and scenes M as columns. Users annotate the scenes of a domain with the value r_{im},

$$r_{im} = \begin{cases} 1, & \text{if scene } m \text{ is annotated as dangerous} \\ 0, & \text{otherwise (scene } m \text{ is annotated as non-dangerous)} \end{cases}$$

.

As explained previously different domains will generate the content of video scenes in different ways in response to external variables. This makes it difficult to predict the labels of scenes in new domains.

(a) Similar trajectories mapped back to video scenes



(b) Content of some buckets in the hash table

Figure 6.2. Discovery of activities in video. (a) Similar trajectories show a similar shape in video scenes. (b) They are mapped to buckets A, B, C, and D of TSH.

6.3 Related Work

6.3.1 Latent Dirichlet Allocation (LDA)

We use LDA to generate topics in video. A topic is a distribution over activities that co-occur frequently in scenes. Every scene is thus formed by a multinomial proportion over K topics.

Let's fix the following parameters of the model: K topics, β (each β_k is a vector of probabilities over the V entries of the dictionary of activities), and the Dirichlet parameter α (a vector of K components with $\alpha_i > 0$). LDA models every scene m with the following generative process:

1. Choose a topic proportion θ_m from the distribution Dirichlet(α)

2. For each of the N activities,

   (a) Choose topic id z_{mn} from the multinomial distribution Mult(θ_m)

   (b) Choose an activity a_{mn} from the multinomial distribution Mult($\beta_{z_{mn}}$)

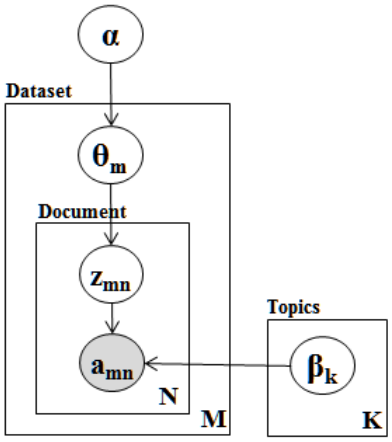


Figure 6.3. Graphical model representation of LDA. Activities are shadowed to represent an observable variable.

The above process explains how LDA allows a scene to exhibit a mixed membership over the K possible topics. For example, LDA can capture that the scene shown in Figure 6.1 (c) the scene contains topics β_1 and β_2, where β_2 corresponds to a collection of two frequently co-occurring activities (vehicles going down and vehicles turning right) in the second lane. A latent variable for this scene represents the proportions over $K = 5$ possible topics showing that topics β_1 and β_2 are active, $\theta = \{0.6, 0.4, 0.0, 0.0, 0.0\}$. Note that similar scenes will exhibit similar topic proportions, θ. Hence, θ_m provides a low-dimensional representation for the content of a video scene, m.

LDA is formed of conditional relationships between activities, topics, and parameters. Figure 6.3 shows these dependencies as a probabilistic graphical model. It includes *hidden variables* (per-activity topic assignments z_{mn}, per-scene topic proportions θ_m, and per-dataset topic distribution β_k), *observable variables* (activities a), and a *dataset-level parameter* (Dirichlet parameter α). The hidden variables reflect a space of latent variables for the corpus, so its posterior inference is needed to estimate which topics best fit to activities in every scene. The posterior distribution over all the random variables in the model can be decomposed into a product of conditional distributions,[1] as follows.

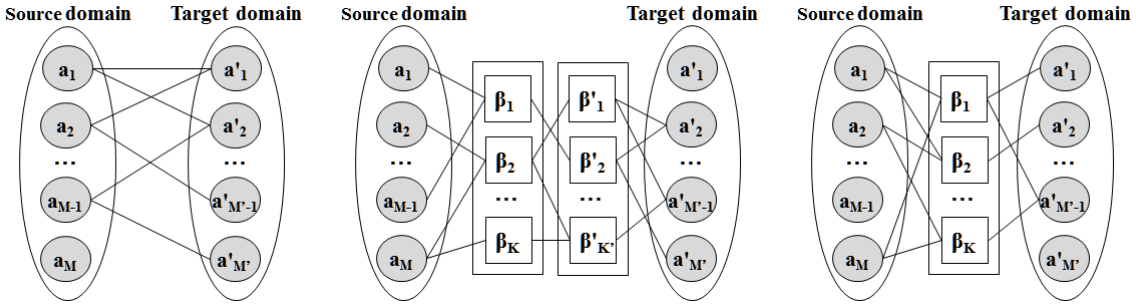
[1]Both Equation (6.1) and Figure 6.3 are equivalent as they represent conditional dependence between random variables

$$p(\theta, z|a, \alpha, \beta) = \prod_{n=1}^{N} p(\theta|\alpha) \prod_{n=1}^{N} p(z_n|\theta)p(a_n|z_n, \beta) \tag{6.1}$$

We can use variational Expectation Maximization (EM) as a deterministic alternative to Markov Chain Monte Carlo (MCMC) to approximate the computation of the posterior distribution $p(\theta_{1:M}, z_{1:M}|a_{1:N}, \alpha_{1:K}, \beta_{1:K})$ as in [51]. This will optimize independently the variational parameters that govern the latent variables of Equation (6.1) by minimizing the Kulllback-Leibler (KL) divergence between the variational distribution and the true posterior $p(\theta, z|w, \alpha, \beta)$.

6.3.2 Activity Mining

The recognition of activities in video is an open problem that has received much attention lately. Commonly, low-level visual features and actions have been modeled and classified to provide interpretation of activities. While the traditional way to categorize existing research is by motion representation such as local features (e.g., changes in velocity, motion trajectories, and gradients) or global features (e.g., key frames), recent research has employed hierarchical Bayesian models such as LDA [51] and Hierarchical Dirichlet Process (HDP) [48] to cluster those activities into scenes [52–54]. The above research has led to techniques that can discover atomic scenes, but such techniques omit the complex interactions between scenes commonly present in video. Wang et al. [54] approach this problem by adding one more level to the hierarchy of the LDA and HDP Bayesian models and providing extended versions of integral probabilistic hierarchical Bayesian models (LDA, HDP, and Dual-HDP mixture models) to cluster moving pixels into atomic activities and interactions. Similarly, Li et at. [56] infer global behavior patterns through modeling behavior correlations through a hierarchical probabilistic Latent Semantic Analysis (pLSA). Both techniques, however, learn global interactions disregarding temporal variations in the distribution of activities. By contrast, our technique models video scenes with topics that do not assume the same probability of co-occurring activities over time, a reasonable scenario imposed by the processing of continuous video streams.



(a) Feature-level (b) Topic-level (c) Cross-domain level

Figure 6.4. Three different strategies for Domain Adaptation. The random variables $a_{1:M}$ and $\beta_{1:K}$ represent activities and latent variables, respectively.

Despite all of the existing research in activity mining in surveillance videos, little has been done on taking advantage of user's annotations of scenes across domains. In this work, we combine content and user annotation in different domains to predict the labels of future scenes.

6.4 Domain Adaptation

The mining of activities for multiple domains exhibits new characteristics to existing algorithms that also model video scenes as a combination of topics, yet in a single domain [52, 54]. Topics get partially preserved when we project them to other domains. In our video problem, this happens because both source and target domains have different vocabularies of activities and different marginal and conditional probabilities (dependency relationships between random variables). These issues violate fundamental assumptions of topic modeling for mining crowd activities in video.

However, the content of two different domains is transferable if both domains share a set of common variables (which can be observable or latent). Our goal is to discover domain-independent variables that connect similar content in two domains. We analyze this transferring at three different levels: feature-level, topic-level, and cross-domain level.

6.4.1 At a feature-level

A simple approach is to find a set of observations (activities) present in the dictionary of both domains, as seen in Figure 6.4 (a) and then perform a K-means clustering over these common features to generate K mixture components. Then, in each scene we obtain a low-dimensional representation on the content by computing a histogram of activities indexed by each of the K clusters and normalizing.

6.4.2 At a topic-level

We compute topics with LDA in each domain independently and define a similarity measure between topics in terms of the number of activities that they share across domains. Every topic is connected to its most similar topic in the other domain, as seen in Figure 6.4(b). When we separate connected topics, we end up with a new set of mixture components that connects content in both domains. As before, we compute the content for each scene by counting the number of activities indexed by each mixture component.

6.4.3 At a crossdomain-level

The last two approaches are sub-optimal solutions to find latent variables that are relevant for domain adaptation. This is because either video activity is noisy and unstable or domain-specific topics model co-occurrence of activities in each domain, but fails to represent mutual co-occurrence across domains.

We improve the above two methods by collapsing activities of both domains in a combined dataset and extracting topics to have a common latent variable to explain both domains, taking special care to consider the same number of activities in each domain to avoid bias in the generation of topics for a particular domain, as shown in Figure 6.4(c). Our experiments show that this method provides better recall values than feature and topic level techniques.

	Source domain$_1$				*Source domain*$_2$				*Source domain*$_3$				*Target domain*			
	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}
$user_1$	1	0	1	0	1	0	1	0	1	0	0	0	X	X	X	X
$user_2$	1	0	1	0	1	0	0	0	0	0	0	0	X	X	X	X
$user_3$	0	0	0	0	1	0	1	0	0	0	0	1	X	X	X	X
$user_4$	1	1	1	0	1	1	1	0	1	0	0	0	X	X	X	X
$user_5$	0	0	1	0	0	0	1	0	1	0	0	0	X	X	X	X

Figure 6.5. Multiple source and target domains, missing annotations are shown with `X`s.

6.5 Crossdomain Probabilistic Model

Consider the setting of three source domains and a current target domain depicted in Figure 6.5. Scenes in source domains are annotated as dangerous ($r_{ij} = 1$) or non-dangerous ($r_{ij} = 0$). Conversely, scenes of the target domain are not annotated ($r_{ij} =$`X`). We would like a model that predicts those missing values based on the similarity of content in previous source domains and in how multiple users similarly annotate scenes as dangerous.

A source domain can be factorized into latent vectors $u_i \in \mathbb{R}^K$ (for users) and $v_m \in \mathbb{R}^K$ (for scenes), respectively. The annotation of user i to scene m in the original matrix can be approximated by the inner product between their corresponding latent vectors,

$$r_{im} \sim u_i^T v_m \tag{6.2}$$

by minimizing the least squared error with respect to true user annotations r_{im},

$$min_{U,V} \sum_{i,m} (r_{im} - u_i^T v_m)^2 + \lambda_u \|u_i\|^2 + \lambda_v \|v_m\|^2 \tag{6.3}$$

with regularization parameters λ_u and λ_v.

Note that the inner product $u_i^T v_m$ corresponds to a prediction of whether user i considers the interaction of activities contained in scene m as dangerous. However, such a prediction cannot be computed in a *target domain* because those scenes are not annotated and therefore r_{im} is not available for Equation 6.2, as visually described in Figure 6.5.

We introduce the Crossdomain Probabilistic Model (CPM) to alleviate that problem. CPM learns user annotations on latent variables that connect the generation of content across a pair of source and target domains (see Section 6.4.3 and Figure 6.4(c)). CPM uses those cross-domain topic proportions, θ_m, in place of the latent vector v_m of Equation (6.2) to factorize the user annotations in both domains, as follows.

$$r_{i,m} \sim \mathcal{N}(u_i^T \theta_m, c_{ij}^{-1}) \tag{6.4}$$

being c_{ij}^{-1} a precision parameter.

However, such predictions should include an uncertain quantity ϵ_m that offsets the topic proportion in response to the dissimilarity of their corresponding domains. This information is encoded using a normal distribution with expected value equal to the number of activities in the target domain and variance equal to the number of different entries in the dictionary of activities of both domains.[2] The idea is to penalize source domains that are different in terms of feature representations. Thus, a latent vector v_m explains a scene m as $v_m = \theta_m + \epsilon_m$, where $\epsilon_m \sim \mathcal{N}(\mu, \Sigma)$ is equivalent to

$$v_m \sim \mathcal{N}(\theta_m + \mu, \Sigma)$$

which models when the document latent vector v_m is close to its topic-proportions θ_m. This makes a user annotation equivalent to,

$$\begin{aligned} \mathrm{E}[r_{i,m}|u_i, \theta_m, \epsilon_m] &\sim \mathcal{N}(u_i^T(\theta_m + \mu), \Sigma)) \\ &\sim \mathcal{N}(u_i^T v_m, \Sigma) \end{aligned} \tag{6.5}$$

This cross-domain factorization can also be generated as a probabilistic graphical model with the following generative process,

[2] A prior for ϵ is considered a purely subjective assessment in the way how a target domain is related to other source domains. Different and more expressive metrics to relate content across domains could be used such as the part of the day or the month when the recording took place.

1. For each of the I users,

   (a) Choose a latent vector u_i from $\mathcal{N}(0, \lambda_u^{-1} I_K)$

2. For each of the C collapsed datasets that combines a target domain with a source domain,

   (a) For each of the M documents,

      i. Choose a topic proportion θ_m from the distribution Dirichlet(α)

      ii. Choose the document offset ϵ_m from $\mathcal{N}(\mu_c, \Sigma_c)$

      iii. Set document latent vector as $v_m = \epsilon_m + \theta_m$

      iv. For each of the N activities,

         A. Choose topic id z_{mn} from Mult(θ_m)

         B. Choose an activity a_{mn} from Mult($\beta_{z_{mn}}$)

      v. For each existing user-scene annotation (i, m),

         A. Choose an annotation $r_{i,m}$ from $\sim \mathcal{N}(u_i^T v_m, c_{ij}^{-1})$

Similar approaches that also model user annotations as a probabilistic matrix factorization over users and items do not consider the presence of multiple domains and therefore lose the advantage of transferring existing labels to a target domain to improve prediction [59, 60].

6.5.1 Learning Parameters

Let's assume the topics $\beta_{1:K}$ are fixed, the posterior distribution of CPM $p(u, v, \theta, z|\Sigma, \lambda_u, \alpha, \beta)$ can be factorized as follows,

$\prod_{i=1}^{I} p(u_i|\lambda_u) \prod_{m=1}^{M} p(r_{i,m}|u_i, v_m) p(v_m|\Sigma, \mu, \theta_m) p(\theta_m|\alpha)$

$\prod_{n=1}^{N} p(z_n|\theta_m) \prod_{n=1}^{N} p(a_{m,n}|z_{m,n}, \beta_k)$

This expression is computationally intractable as the cardinality $VxMxN$ is huge. Our goal is to maximize the posterior distribution by optimizing the functional dependence of

the Gaussian on each hidden parameter. For example, the posterior of $p(v_m|\mu, \Sigma, \theta_m)$ has the quadratic form,

$$(v_m - (\theta_m + \mu_m))^T \Sigma_m (v_m - (\theta_m + \mu_m))$$

because each topic proportion is biased in response to a dissimilarity metric encoded in the covariance matrix Σ, $v_m \sim \mathcal{N}(\theta_m + \mu, \Sigma)$. Similarly, the quadratic form of the user annotation $p(r_{i,m}|u_i, v_m)$ is defined as,

$$(r_{i,m} - (u_i^T v_m))^T c_{i,j} (r_{i,m} - (u_i^T v_m))$$

The log likelihood of the posterior represents this value as the sum of probabilistic components,

$$\begin{aligned}
&\mathcal{L} = -\tfrac{\lambda_u}{2} \textstyle\sum_i u_i^T u_i - \tfrac{1}{2} \sum_m (v_m - (\theta_m + \mu_m))^T \Sigma (v_m - (\theta_m + \mu_m)) \\
&- \textstyle\sum_{i,j} (r_{i,m} - \tfrac{1}{2}(u_i^T v_m))^T c_{i,j} (r_{i,m} - (u_i^T v_m)) \\
&+ \textstyle\sum_m \sum_n log(\theta_{jk} \beta_{k,w_{jn}})
\end{aligned}$$

First, we optimize $\mathcal{L}$ in terms of U and V and set them to zero to find their optimal parameters, similar to [61]. Then, we derive $\mathcal{L}$ in terms of θ to learn the topic proportions for each document and apply Jensen's inequality to provide a lower bound in terms of a function $q(\theta)$ that can be factorized with K independent components as $q(\theta) = \prod_i^K q(\theta_i)$. The result the following factorization, $q(\theta) = q_1(\theta_{mk}) q_2(\beta_{k,w_{mn}})$, which can be optimized by coordinate ascend to find their optimum parameters by fixing one of them at each iteration.

6.6 Experiments

In this section, we test the performance of CPM with surveillance videos recording activities in traffic roads. Moving objects describe traffic scenes governed by the state of multiple semaphores. We experiment on the following datasets: ***Street Intersection***[3]

[3]http://www.eecs.qmul.ac.uk/∼jianli/Junction.html

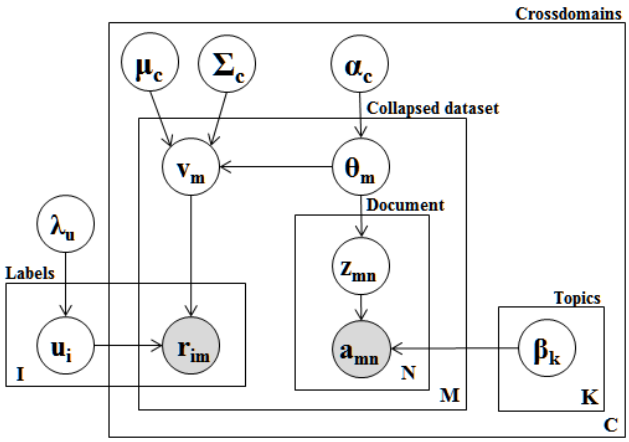


Figure 6.6. Probabilistic Graphical Model of CPM. Activities a_{mn} and annotations r_{im} are observable variables shadowed in gray. Note that domain independent topic proportions θ_m are defined for a each collapse dataset.

(*normal quality, 25fps, 90 minutes, 5 semaphores*), ***Karl-Wilhelm*** & ***Strabe Streets***[4] (*normal definition, 25fps, 2 hour, 3 semaphores*), and ***Roundabout Junction***[5] (*normal quality, 25fps, 2 hour, 3 semaphores*).

We generate a ground truth in every dataset by asking users to manually tag time windows of 10 seconds with either "dangerous" or "safe". We generate domains in each dataset by considering three consecutive source domains of 20 minutes at the beginning of each video and a target domain of 20 minutes in the end of the video. This generates 360 scenes for the source domain and 120 scenes for the target domain. To compute precision, we assume that no labels are available in the target domain and consider a hit if the average value of the precision with respect to the number of users is larger than 0.5 and a miss otherwise. For the generation of dictionaries of activities in each video, we use the same hash function in Timeseries Sensitive Hashing (TSH). The number of activities found in each video depends on the complexity of the scene, as shown in Figure 6.1. The more traffic lights yields more segmented activities and a larger dictionary. The experiments are run on a 3.6 GHz Pentium 4 with 4 GB RAM and all the above datasets are publicly available to

[4]http://i21www.ira.uka.de/image_sequences/
[5]http://www.eecs.qmul.ac.uk/~jianli/Roundabout.html

Table 6.1. Information of the Training of Each Dataset.

Dataset	**# Activities**	**Time to Compute**
Street Intersection (5 semaphores)	318	4.21 hours
Karl-Wilhelm & Strabe (3 semaphores)	227	4.37 hours
Roundabout Junction (3 semaphores)	235	4.43 hours

facilitate later experimental comparisons.

6.6.1 Experiment 1: Finding Parameters

Before studying the performance of the algorithm, we analyze in this experiment the optimal values of external variables K (number of topics) and Σ (covariance between a source and target domain) for each dataset considering all the datasets.

First, we want to study the effect of changing the numbers of topics K in the model. We fix the other input parameters $\{\alpha = 0.1,\ \lambda_u = 0.5,\ \Sigma = 0.3, 0.3, 0.3\}$ and then compute the Mean Average Precision (MAP@20) for 30% of the scenes in the target domain with a varying number of topics. Figure 6.7 shows that a value of $K = 50$ topics provides a consistent result for all the datasets. Note that, the Street Intersection dataset shows the highest precision as it also contains the largest total number of activities. This is because scenes are better defined by topics that model co-occurrence with enough number of discrete activities.

Second, we observe the effect of the covariance matrix to relate different combinations of source and target domain in a video. We choose the Roundabout Junction dataset for this experiment because it is the largest, so it is more likely to find multiple domains. We fix this time the number of topics to $K = 50$, in response to the above experiment, and the other input parameters as $\{\alpha = 0.1,\ \lambda_u = 0.5\}$. Then we compute the Mean Average Precision (MAP@20) for 30% of the scenes in the target domain, but considering each of the three source domains independently. Only one component of the covariance matrix is activated at a time to distinguish its entire effect (e.g., when considering the first source domain, only $\Sigma_{i=t,j=1}$ is activated). Figure 6.8 shows that the last domain gets the highest

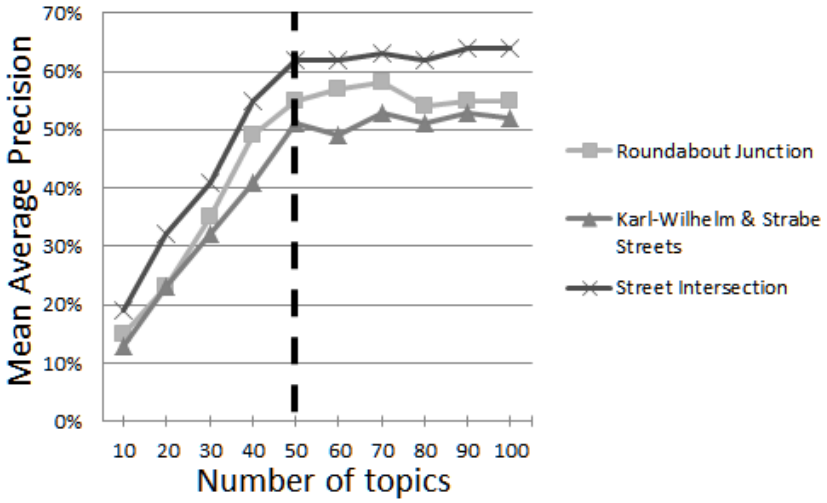


Figure 6.7. Finding the optimal number of topics for each videoset in CPM.

precision with a value of $\Sigma_{i=t,j=3} = 0.8$. The choice of the last domain makes sense as it is the closest in time and thus the most likely to contain similar distributions in latent variables. Note that this effect may vary in much larger videos.

6.6.2 Experiment 2: Average Prediction in Target Domain

To make a competitive comparison, we evaluate the performance of CPM with the optimal K and $Sigma$ parameters computed in Experiment 1 and $\{\alpha = 0.1, \lambda_u = 0.5\}$, in terms of the Mean Average Precision (P@20) for predicting the true label in the target domain. This experiment examines the following methods:

a) SVM (trained with common feature-level vectors, as explains in Section 6.4.1)

b) SVM (trained with common topic-level vectors, as explains in Section 6.4.2)

c) Collaborative Filtering (concatenating both source and target domains in a single matrix) and

d) CPM

As summarized in Figure 6.9, CPM shows better prediction of dangerous scenes in the target domain. The closest competitor is Collaborative Filtering, which does not consider

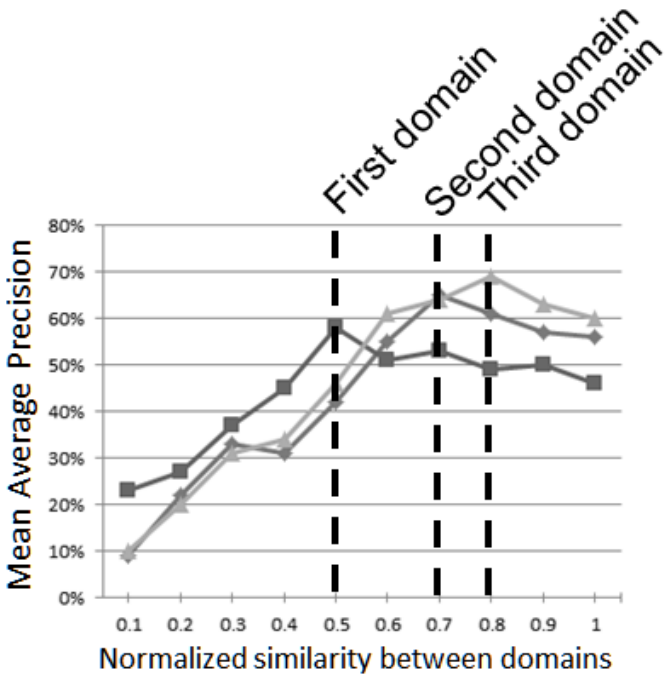


Figure 6.8. Finding the optimal values in the covariance matrix that relates content with respect to each source domain.

any video content, but only user preferences on scenes. This shows the advantage of combining content and user annotations for describing activities in surveillance video. On the other hand, SVM cannot make adequate generalization as the underlying distributions of observations and latent variables change for different domains. This is more evident when the temporal gap between source and target domains is large as in the Karl-Wilhelm & Strabe Streets and Roundabout Junction datasets. However, when the temporal distance between source domains and the target domain is small, the performance of SVM with topic-level vectors is similar to Collaborative Filtering.

6.7 Conclusion

In this chapter, we have discussed methods to transfer labels from an existing domain to a target one in a surveillance video. This transferring happens when two domains share a set of random variables. In this chapter, we have studied the way how crossdomain topics have

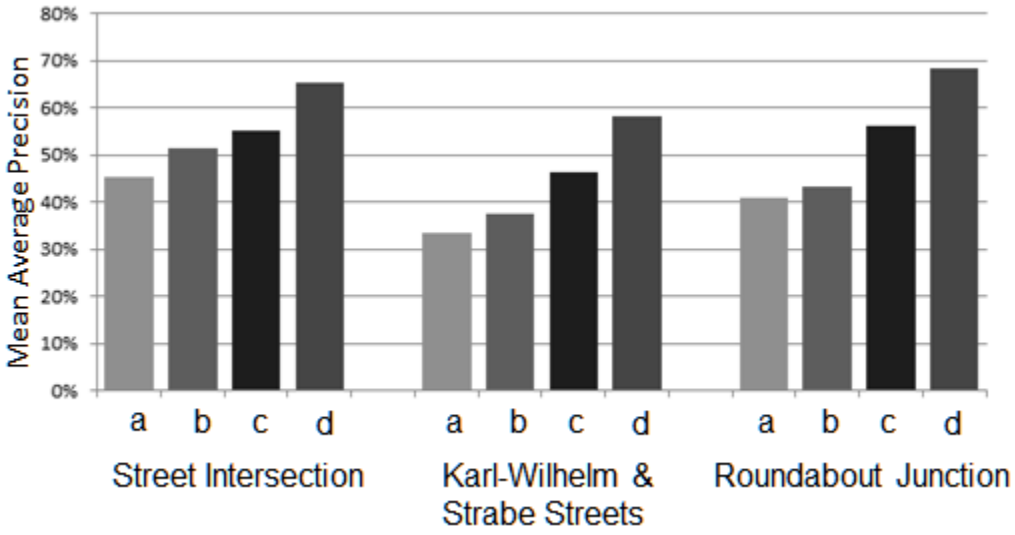


Figure 6.9. Comparison of prediction techniques in terms of Mean Average Precision. (a) SVM (trained with common feature-level vectors) (b) SVM (trained with common topic-level vectors) (c) Collaborative Filtering and (d) CPM

the ability to bridge domain with co-occurrence of activities. Finally, we let users guide the relevance of topics to explain the presence of dangerous activities in a collaborative basis.

CHAPTER 7

CONCLUSIONS

7.1 Introduction

Video is a massive amount of data that contains complex interactions between moving objects. Moreover, the continuous recording of activities during long periods is needed to capture interesting events. We then need different techniques to analyze video to uncover the relationships between moving objects and to learn the correspondence between content and labels. For example, a surveillance camera recording traffic scenes will require methods to extract patterns from moving objects (e.g., trajectories). As that information is continuous and affected by noise, we will need to discretisize it into a dictionary of representative activities. A scene is thus defined by a subset of activities in a time window. How some activities co-occur is dynamic and can be explained by a statistical mixture of distributions over activities in multiple time windows.

The research presented in this dissertation provides the basis for analyzing video. The primary contributions of this research consist of providing new algorithms for (a) the efficient search of activities in video, (b) scene understanding based on interactions between activities, and (c) the predicting of labels for new scenes. The results from this research provide techniques to achieve those goals. In the following sections, we address our contributions to these areas as well as possible extensions and future work.

7.2 Contributions

The algorithms presented in this dissertation have been tested by comparing their performance to existing approaches in the state-of-the-art. The superior performance in the experiments supports the hypothesis stated in each chapter in relation to the analysis

video. We identify the following contributions in each of the areas above mentioned.

7.2.1 Contributions to Similarity Search

Consider trajectories in video that represent spatio-temporal behaviors of moving objects. We proposed a hashing algorithm, Timeseries Sensitive Hashing (TSH), which generalizes the scalar projection (dot product) to support the indexing of timeseries of variable length. We computed the optimal parameters of the data structure in a data-driven approach to efficiently index human motion timeseries that come from non-uniform distributions [4,5] and also activities of vehicles in traffic videos [8]. The initial hypothesis was that the activities in video could be indexed with better precision and faster time response if we would consider a hashing approach that encoded a non-linear correspondence of trajectories within its hash function. This conjecture was more general than previous approaches as the indexing of vectors of same dimensionality became a special case.

The results of the experiments provided evidence that supported that hypothesis. For example, TSH was shown to have better average precision when performing 1-nearest neighbor search in three different motion databases (CMU motion capture database, HDM05 mocap database, and Human Motion Video dataset). The advantage in terms of average precision for using TSH is 31.7% in comparison to tree-based approaches (M-Tree and R-tree) and 23.5% when TSH was compared to another hash approach (LSH). In contrast to the other competitors, TSH does not need to normalize the timeseries in the dataset to the same dimension because it encodes the Dynamic Time Warping (DTW) distance in the hash function. TSH thus takes advantage of the original information contained in the data to distinguish non-similar timeseries and provides better precision values.

When we study the time response of TSH by incrementally increasing the size of the dataset from 10% to 100% of the original size, TSH exhibited good scalability, having LSH as the closest contender. This is because both algorithms are good at distributing timeseries into buckets uniformly and quickly retrieving them using hashing functions. However, in contrast to LSH, TSH is specially designed to reduce the number of collisions for timeseries of variable length while providing good precision results.

The code that implements TSH was released in October, 2012 under the MIT License and it is free for download and modification [62].

7.2.2 Contributions to Scene Understanding

We provided a method to find approximate co-occurring associations from a video stream by considering the unsupervised clustering of discrete events. The frequent co-occurring relationships between activities are incrementally computed over the stream [7,63]. Our main hypothesis during this investigation was that video scenes could be modeled by the ways in which words group into topics in a text document. Every scene in the video thus corresponded to a multinomial distribution over a number of independent topics. This is similar to how words form topics in a text document. Most previous research had considered the generation of topic models for large text datasets. Our work extended those findings to a noisier domain such as video databases and demonstrated that a hierarchy of random variables could automate the discovery of scenes in video.

The experiments consisted of discovering interactions of activities based on association rules. The generated rules helped us to understand the temporal co-occurring relationships between activities in three real traffic video datasets (Street Intersection, Karl-Wilhelm & Strabe, and Roundabout Junction). For example, in the Roundabout Junction dataset, we show how the most frequent rule segmented motion of vehicles into multiple activities such as vehicles going straight and then joining the roundabout. The second most frequent rule explained the co-occurring relationship of vehicles taking lanes separated by the roundabout. While some vehicles circulate alongside the roundabout emerging as two activities, this rule also showed that another set of cars takes a different way by turning left from the center to the upper part of the scene. A visual inspection of these rules supported the original hypothesis considering topics as reliable components to describe the content of video scenes.

7.2.3 Contributions to Scene Prediction

We introduced a statistical model to approximate the value of random variables. Their dependency explains the co-occurrence of activities and annotations in different parts of a

video. The random variables for the content correspond to the topics that we discussed for scene understanding. The random variable for the annotations describes how users annotate the content in video with labels. Our hypothesis was that the topics that describe scenes are stable enough to be present in the video. Thus, they could be used to transfer labels from one part of the video to another.

We implemented a Probabilistic Graphical Model, Crossdomain Probabilistic Model (CPM) [9, 10], that automatically found the topics that model the co-occurrence of trajectories in two parts (domains) of a video. Experiments supported our hypothesis by showing improvement in the average precision of predicting labels for new scenes when we use 20 minutes of previous content located at the beginning of each video. We experimented with implementations of the Support Vector Machine (SVM) that consider topic modeling scenes either in a source domain or in a cross domain representation, but the closest competitor was Collaborative Filtering. This showed the power of predicting labels by considering how other people labeled video scenes. CPM demonstrated that the idea of incorporating latent variables that explained content across domains, and then finding the linear correspondence of those to random vectors associated to user annotations on previous scenes, increases the average precision by 15% in the prediction of labels for new scenes. Those results thus gave evidence to support the initial hypothesis.

7.3 Future Work

The models and techniques proposed in this work are only initial steps to solve the problem of video analysis and much work remains. Specifically, some characteristics that are not considered yet include the incremental and distributed learning of models for prediction. The current models can handle large videos, but do not yet enable the kind of real-time updating to deal with streaming scenarios in security and surveillance or the possibility of obtaining annotations from an online community via crowdsourcing. This section offers an overview of research necessary to move our models and methods in that direction.

7.3.1 Future Work on Similarity Search

Previous research on efficiently indexing timeseries typically relies on dimensionality reduction to eliminate variable dimensionality. Current practice is to employ dimensionality reduction techniques (iSAX [34], PAA [31], Uniform Scaling [29], etc.) to make timeseries more tractable to compute. The intuition is that dimension reduction preserves enough information to quickly discard non-similar timeseries in a search. These indexing techniques fall into two categories: region-based and lexicographic. R-tree is a region-based index commonly used for experiments with timeseries and one example of a lexicographic index is TS-tree [31], which avoids subtree overlap during insertion. There are (at least) two clustering techniques that employ a hashing approach [42,43]. However, all those techniques only consider the case where points have the same dimensionality and Euclidean distance is the only option as a similarity measure. TSH is especially suitable in scenarios of stream data processing such as motion capture, speech recognition, and sensor networks, where timeseries continuously arrive with different lengths, preprocessing steps are not always possible, and low error rates are required.

Future work on similarity search will consider the problem of storing the search index in secondary memory and representing it in a distributed environment. This feature responds to the presence of a massive amount of trajectories that cannot fit into main memory. The distributed nature of TSH, mapping of elements into independent buckets in constant time, is adequate for being implemented under the MapReduce programming model [64]. Thus, a Map operation will project a timeseries to a specific bucket in a distributed node and a Reduce operation will collect and store in hard disk all the timeseries that were mapped to the same bucked, but in distributed nodes. The implementation of TSH in a distributed environment will reduce the computational time of clustering algorithm for massive timeseries datasets. This is because every bucket in TSH already contains a group of similar elements after indexing all the elements of the dataset.

7.3.2 Future Work on Scene Understanding

Related efforts to model activities in video consider co-occurring relationships between

low-level visual features. Recent research employs Bayesian models such as HDP [48] and LDA [51] to successfully cluster discrete activities into topics [52–54]. Those methods lead to techniques that can discover atomic components to represent scenes, but they omit the complex interactions between topics. Wang et al. [54] approached this problem by adding one more level to the hierarchy of LDA to thus describe global behavior patterns through topic correlations. By contrast, our on-line technique assumes a fixed number of topics and relates their frequent co-occurrence in multiple time windows. In other words, by decoupling both the discovery of activities and interactions, we can incrementally learn interactions without assuming the same probability of co-occurring relationships over time. This is a reasonable scenario imposed by the processing of continuous video streams, where the definition of interactions between activities is more dynamic than the definition of activities in itself.

Future work on scene understanding will need to consider the problem of finding topics incrementally. Faster methods to infer over random variables are needed to detect the emergence of new subsets of activities that frequently co-occur. The Collapsed Variational Bayes (CVB) method is specially designed to approximate inference for a probabilistic graphical model by assuming that latent random variables are mutually exclusive [65]. CVB has been shown to be more computationally efficient than the standard variational Bayes inference algorithm described in Chapter 6. However, CVB still cannot perform incremental updates for topic discovery.

7.3.3 Future Work on Scene Prediction

Previous attempts to predict labels for new scenes in video assume that the distribution of activities is the same along the video [52, 54]. We consider that different parts in the video define individual domains, where the vocabularies of activities and conditional probabilities between random variables are particular to each domain. Existing research performs prediction by considering a single domain [57, 58]. This leads to poor precision and recall when marginal and conditional distributions change in the video. Despite all of the existing research in activity mining in surveillance videos, little has been done on taking

advantage of users' annotations of scenes across domains. In our work we consider that the content of two different domains is transferable if both domains share a set of common variables (which can be observable or latent). Our goal is to discover domain-independent variables that connect similar content in two domains.

Future work on the prediction of labels for video scenes will study how labels are generated to explain interactions of activities. Note that those labels are usually provided by a small group of users that may be biased toward some events expected in the video recording (e.g., dangerous scenes). Crowdsourcing is a technique that uses a large online community to subdivide repeating tasks that can be solvable by human interpretation. For example, we could ask people to annotate scenes as dangerous or safe according to their common sense and choosing the right labels by voting. This will provide more reliable labels for scenes and the prediction of labels for new scenes will provide more meaningful results as well. Existing platforms for performing crowdsourcing with a real online community include Amazon's Mechanical Turk [66] and Microwork [67].

REFERENCES

[1] YouTube, "YouTube statistics," http://www.youtube.com/yt/press/statistics.html/, 2013, [Online; accessed March 03, 2013].

[2] GovWin, "Big data, cloud and video driving increased demand for federal electronic data storage market," http://govwin.com/bernaspi_blog/big-data-cloud-and-video/713070, 2012, [Online; accessed March 03, 2013].

[3] C. J. Galbán, M. K. Han, J. L. Boes, K. A. Chughtai, C. R. Meyer, T. D. Johnson, S. Galbán, A. Rehemtulla, E. A. Kazerooni, F. J. Martinez, and B. D. Ross, "Computed tomography-based biomarker provides unique signature for diagnosis of copd phenotypes and disease progression," *Nature Medicine*, vol. 18, pp. 1711–1715, 2012.

[4] O. U. Florez, A. Ocsa, and C. Dyreson, "Sublinear querying of realistic timeseries and its application to human motion," *Proceedings of the International Conference on Multimedia Information Retrieval*. ACM, 2010, pp. 137–146.

[5] O. U. Florez and C. Dyreson, "Scalable similarity search of timeseries with variable dimensionality," *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, 2011, pp. 2545–2548.

[6] O. U. Florez and S. Lim, "HRG: A graph structure for fast similarity search in metric spaces," *Proceedings of the Database and Expert Systems Applications (DEXA)*. Springer, 2008, pp. 57–64.

[7] O. U. Florez and C. Dyreson, "Discovering activity interactions in a single pass over a video stream," *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 2012, pp. 819–826.

[8] O. U. Florez and C. Dyreson, "One-scan rule extraction to explain significant vehicle interactions with guaranteed error value," *Proceedings of the ACM SIGAPP Applied Computing Review*, vol. 12, no. 2, 2012, pp. 27–38.

[9] O. U. Florez and C. Dyreson, "Is that scene dangerous?: transferring knowledge over a video stream," *Proceedings of the Ph.D. Workshop in the CIKM Conference (CIKM 2012)*, vol. 1, no. 1, 2012, pp. 23–28.

[10] O. U. Florez and C. Dyreson, "What to reuse?: A probabilistic model to transfer user annotations in a surveillance video," submitted to the *Proceedings of 19th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2013.

[11] E. Keogh, L. Wei, X. Xi, S.-H. Lee, and M. Vlachos, "LB_keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures," in *VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases*. VLDB Endowment, 2006, pp. 882–893.

[12] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana, "Fast time series classification using numerosity reduction," in *ICML '06: Proceedings of the 23rd International Conference on Machine Learning*, 2006, pp. 1033–1040.

[13] A. Fu, E. Keogh, L. Y. H. Lau, and C. A. Ratanamahatana, "Scaling and time warping in time series querying," in *VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB Endowment, 2005, pp. 649–660.

[14] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan, "Detecting time series motifs under uniform scaling," in *KDD '07: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007, pp. 844–853.

[15] D. Minnen, T. Starner, I. Essa, and C. Isbell, "Discovering characteristic actions from on-body sensor data," *10th IEEE International Symposium on Wearable Computers*, pp. 11–18, Oct. 2006.

[16] A. G. Kirk, J. F. O'Brien, and D. A. Forsyth, "Skeletal parameter estimation from optical motion capture data," in *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, 2005, pp. 782–788.

[17] J. F. Knight, H. W. Bristow, S. Anastopoulou, C. Baber, A. Schwirtz, and T. N. Arvanitis, "Uses of accelerometer data collected from a wearable system," *Personal Ubiquitous Computing*, vol. 11, no. 2, pp. 117–132, 2007.

[18] E. Hsu, K. Pulli, and J. Popović, "Style translation for human motion," *ACM Transaction on Graphics*, vol. 24, no. 3, pp. 1082–1089, 2005.

[19] E. Keogh, B. Celly, C. A. Ratanamahatana, and V. B. Zordan, "A novel technique for indexing video surveillance data," in *IWVS '03: First ACM SIGMM International Workshop on Video Surveillance*, 2003, pp. 98–106.

[20] V. Kobla, D. Doermann, and C. Faloutsos, "Videotrails: Representing and visualizing structure in video sequences," *Proceedings of ACM Multimedia*, 1997, pp. 335–346.

[21] M. Fleischman, P. Decamp, and D. Roy, "Mining temporal patterns of movement for video content classification," in *MIR '06: Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval*, 2006, pp. 183–192.

[22] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local svm approach," in *ICPR '04: Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04) Volume 3*, 2004, pp. 32–36.

[23] C. Lai, T. Rafa, and D. E. Nelson, "Mining motion patterns using color motion map clustering," *SIGKDD Explorations Newsletter*, vol. 8, no. 2, pp. 3–10, 2006.

[24] I. Laptev and T. Lindeberg, "Space-time interest points," in *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, 2003, p. 432.

[25] J. Yan and M. Pollefeys, "Video synchronization via space-time interest point distribution," *Proceedings of Advanced Concepts for Intelligent Vision Systems*, 2004.

[26] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey Vision Conference*, 1988, pp. 147–152.

[27] E. M. M. Swe, M. Pwint, and F. Sattar, "On the discrimination of speech/music using a time series regularity," in *ISM '08: Proceedings of the 2008 Tenth IEEE International Symposium on Multimedia*, 2008, pp. 53–60.

[28] T. Schreck, T. Tekušová, J. Kohlhammer, and D. Fellner, "Trajectory-based visual analysis of large financial time series data," *ACM SIGKDD Explorations Newsletter*, vol. 9, no. 2, pp. 30–37, 2007.

[29] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and Information Systems*, vol. 7, no. 3, pp. 358–386, 2005.

[30] Y. Zhu and D. Shasha, "Warping indexes with envelope transforms for query by humming," in *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 2003, pp. 181–192.

[31] I. Assent, R. Krieger, F. Afschari, and T. Seidl, "The TS-tree: efficient time series search and retrieval," in *11th Conference on Extending Database Technology (EDBT)*, 2008, pp. 252–263.

[32] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *SCG '04: Proceedings of the Twentieth Annual Symposium on Computational Geometry*, 2004, pp. 253–262.

[33] A. Fu, E. Keogh, L. Y. H. Lau, and C. A. Ratanamahatana, "Scaling and time warping in time series querying," in *VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases*. VLDB Endowment, 2005, pp. 649–660.

[34] J. Shieh and E. Keogh, "iSAX: indexing and mining terabyte sized time series," in *KDD '08: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008, pp. 623–631.

[35] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 518–529.

[36] Z. Yang, W. T. Ooi, and Q. Sun, "Hierarchical, non-uniform locality sensitive hashing and its application to video identification," in *ICME '04: IEEE International Conference on Multimedia*, 2004, pp. 743–746.

[37] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD Conference*, 1984, pp. 47–57.

[38] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 426–435.

[39] E. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle, "Indexing large human-motion databases," in *VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases*. VLDB Endowment, 2004, pp. 780–791.

[40] O. U. Florez and S. Lim, "Discovery of interpretable time series in video data through distribution of spatiotemporal gradients," *Proceedings of the 24th Annual ACM Symposium on Applied Computing (ACM SAC 2009)*, 2009, pp. 112–120.

[41] BerkleyDB, "BerkleyDB Turk," http://www.oracle.com/database/berkeleydb/, 2013, [Online; accessed March 03, 2013].

[42] H. Koga, T. Ishibashi, and T. Watanabe, "Fast agglomerative hierarchical clustering algorithm using locality-sensitive hashing," *Knowledge and Information Systems*, vol. 12, pp. 25–53, May 2007.

[43] Y.-P. Wu, J.-J. Guo, and X.-J. Zhang, "A linear DBScan algorithm based on LSH," *6th Conference on Machine Learning and Cybernetics,*, pp. 115–125, August 2007.

[44] C.-S. Leung and Q. Khan, "DSTree: A tree structure for the mining of frequent sets from data streams," in *International Conference on Data Mining*, 2006, pp. 928 –932.

[45] Y. Shomura, Y. Watanabe, and N. Ikeda, "A traffic monitoring method for high speed networks," in *International Symposium on Applications and the Internet*, 2009, pp. 107–113.

[46] I. Laptev, M. Marszałek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *Conference on Computer Vision & Pattern Recognition*, June 2008, pp. 311–321.

[47] J. Sethuraman, "A constructive definition of Dirichlet priors," *Statistica Sinica*, vol. 2, no. 476, pp. 639–650, 1994.

[48] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Hierarchical Dirichlet Processes," *Journal of the American Statistical Association*, vol. 101, pp. 1566–1581, 2006.

[49] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *20th International Conference on Very Large Data Bases*, 1994, pp. 478–499.

[50] H. J. Woo and W. S. Lee, "EstMax: Tracing maximal frequent item sets instantly over online transactional data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 10, pp. 1418 –1431, October 2009.

[51] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, March 2003.

[52] T. Hospedales, S. Gong, and T. Xiang, "A markov clustering topic model for mining behaviour in video," in *International Conference on Computer Vision*, 2009, pp. 1165–1172.

[53] J. Yin and Y. Meng, "Human activity recognition in video using a hierarchical probabilistic latent model," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2010, pp. 15–20.

[54] X. Wang, X. Ma, and W. E. L. Grimson, "Unsupervised activity perception in crowded and complicated scenes using hierarchical bayesian models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 3, pp. 539–555, 2009.

[55] Y. Wang and G. Mori, "Human action recognition by semi-latent topic models," *IEEE Transactions on Pattern Analysis and Machine Intelligence Special Issue on Probabilistic Graphical Models in Computer Vision*, vol. 31, no. 10, pp. 1762–1774, 2009.

[56] J. Li, S. Gong, and T. Xiang, "Global behaviour inference using probabilistic latent semantic analysis," in *British Machine Vision Conference*, 2008, 5, pp. 253–265.

[57] O. A. B. Penatti, L. T. Li, J. Almeida, and R. da S. Torres, "A visual approach for video geocoding using bag-of-scenes," *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, ser. ICMR '12, 2012, pp. 1–8.

[58] R. Yonetani, "Modeling video viewing behaviors for viewer state estimation," *Proceedings of the 20th ACM International Conference on Multimedia*, 2012, pp. 1393–1396.

[59] H. Shan and A. Banerjee, "Generalized probabilistic matrix factorizations for collaborative filtering," in *ICDM: International Conference on Data Mining*, 2010, pp. 1025–1030.

[60] C. Wang and D. M. Blei, "Collaborative topic modeling for recommending scientific articles," *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11, 2011, pp. 448–456.

[61] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ser. ICDM '08, 2008, pp. 263–272.

[62] O. U. Florez, A. Ocsa, and C. Dyreson, "Timeseries Sensitive Hashing," http://code.google.com/p/timeseries-sensitive-hashing/, 2013, [Online; accessed March 03, 2013].

[63] O. U. Florez and C. Dyreson, "Mining rules to explain activities in videos," *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, ser. CIKM '10, 2010, pp. 1577–1580.

[64] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[65] Y. W. Teh, D. Newman, and M. Welling, "A collapsed variational bayesian inference algorithm for latent dirichlet allocation," in *NIPS: Neural Information Processing Systems*, p. 2006.

[66] MechanicalTurk, "Mechanical Turk," https://www.mturk.com/, 2013, [Online; accessed March 03, 2013].

[67] Microwork, "Microworkers," http://microworkers.com/, 2013, [Online; accessed March 03, 2013].

[68] C. Traina, A. Traina, B. Seeger, and C. Faloutsos, "Slim-trees: High performance metric trees minimizing overlap between nodes," in *EDBT: Extending Database Technology*, vol. 1777. Springer, 2000, pp. 51–65.

[69] M. R. Vieira, C. T. Jr., F. J. T. Chino, and A. J. M. Traina, "Dbm-tree: A dynamic metric access method sensitive to local density data," in *Brazilian Symposium on Databases*, 2004, pp. 163–177.

[70] G. Navarro, "Searching in metric spaces by spatial approximation," *The VLDB Journal*, vol. 11, no. 1, pp. 28–46, 2002.

[71] H. Hacid and T. Yoshida, "Incremental neighborhood graphs construction for multidimensional databases indexing," in *Advances in Artificial Intelligence*, 2007, pp. 405–416.

[72] H. Hacid and A. D. Zighed, "An effective method for locally neighborhood graphs updating," *Proceedings of the International Conference on Database and Expert Systems Applications*, 2005, pp. 930–939.

[73] D. Zhao and L. Yang, "Incremental construction of neighborhood graphs for nonlinear dimensionality reduction," *Proceedings of the 18th International Conference on Pattern Recognition*, 2006, pp. 177–180.

[74] C. Lee, D. Kim, H. Shin, and D.-S. Kim, "Efficient computation of elliptic gabriel graph," in *International Conference on Computational Science and Applications*, 2006, pp. 440–448.

[75] J. Jaromczyk and G. Toussaint, "Relative neighborhood graphs and their relatives," *General Proceedings of IEEE*, vol. 80, pp. 1502–1517, 1992.

[76] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity search - The metric space approach.* Springer, 2006, vol. 32, no. XVIII.

[77] K. L. Clarkson, "Nearest neighbor queries in metric spaces," in *STOC '97: Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing.* ACM, 1997, pp. 609–617.

[78] G. R. Hjaltason and H. Samet, "Index-driven similarity search in metric spaces (survey article)," *ACM Transactions on Database Systems*, vol. 28, no. 4, pp. 517–580, 2003.

[79] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín, "Searching in metric spaces," *ACM Computing Surveys*, vol. 33, no. 3, pp. 273–321, 2001.

[80] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1993.

[81] S. Brin, "Near neighbor search in large metric spaces," in *VLDB 1995: Proceedings of the 21th International Conference on Very Large Data Bases.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 574–584.

[82] C. Böhm, S. Berchtold, and D. A. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322–373, 2001.

APPENDIX

CHAPTER A

An Incremental Graph Data Structure to Perform Similarity Search for Euclidean Vectors

A.1 Introduction

Searching is a fundamental problem in a modern information system. Indexing is the most well-known technique to speed up a search process, and user queries are evaluated for the given selection predicate over the indexed key by exact matching. However, exact matching may not be suitable for complex and less-structured data objects. For example, the probability for two iris patterns to exactly match is very low, even when they belong to the same individual. For this type of data, imprecise search or search with error, based on the notion of *similarity* (or *dissimilarity*) between objects, has a more practical value.

Similarity search in metric spaces involves distance computation between the data objects. Due to the potential high cost of distance computation for every pair of data objects in a large database, reduction in distance computation becomes the focal point in developing an efficient similarity search technique.

In metric spaces, tree- and graph-based techniques [38,68–71] have been proposed to reduce the search space and subsequently reduce the overall number of distance computations by virtue of the "triangle inequality." However, a tree structure has an inherent limitation: the search starts only from the root of the tree, and once it fails to find a reasonable solution to the given query in the current subtree, backtracking may be unavoidable. Due to this limitation, the authors in [70] noted that a fully connected graph will be an ideal structure for a fast similarity search as long as we can afford the cost for the update of the extra edges. Although reduction in distance computation in graph-based solutions to the similarity search problem has been achieved to a certain extent recently, the intrinsic cost

overhead in graph construction still makes graphs less competitive than trees as a practical solution, and hence the graph-based similarity search in metric spaces has been noted as an open research problem [71–74].

In response to the need of an efficient graph-based solution which takes advantage of no backtracking while avoiding extraneous graph construction cost, we propose in this paper a new efficient, graph data structure, called *Hyperspherical Region Graph* (HRG), for the similarity search problem in a metric space consisted of a large volume of data objects, inspired by the *relative neighborhood graph* (RNG) [75]. The issue of high construction cost of the graph with a large number of data objects is mitigated by modeling the search space as an RNG of vertices each of which represents a set of data objects called *hyperspherical region*, instead of a graph of individual data objects, in our approach. In other words, the proposed graph has substantially less number of vertices than the existing neighborhood graphs, by which efficient graph construction and reduction in distance computation are achieved.

Our empirical analysis of the proposed HRG in comparison with the graph-based data structure (Incremental-RNG) recently proposed in [71] and two state-of-the-art tree-based structures M-tree [38] and SA-tree [70] shows that HRG always outperforms Incremental-RNG in any case with regard to i) data structure construction time and ii) similarity search performance, both of which are tested by the number of distance computations, response time and memory usage. On the other hand, HRG's performance gain over M-tree and SA-tree was noticeable during search time but not in construction time, which is anticipated for a graph structure compared to a tree. HRG performed better than the two tree indices with the 2D, 3D and 16D datasets but not with the 175D dataset used.

The paper is organized as follows: Section A.2 presents a review of previous work. Section A.3 describes the proposed technique. Section A.4 shows the experimental results. Finally, we provide a concluding remark in Section A.5.

A.2 Preliminary Concepts and Previous Work

There have been a number of metric access methods proposed for the search problem in

general metric spaces. Comprehensive surveys on this topic can be found in [76], [77], [78], and [79].

Metric space:. Formally, given a universe $\mathbb{U}$ of data objects and a distance function $d : \mathbb{U} \times \mathbb{U} \rightarrow \mathbf{R}^+$, a subset $\mathbb{S}$ of $\mathbb{U}$ is called a *metric space* if d satisfies the following axioms for any objects x, y, z in $\mathbb{S}$: (1) $d(x, y) \geq 0$, (2) $d(x, y) = d(y, x)$, and (3) $d(x, y) \leq d(x, z) + d(y, z)$ (triangle inequality).

The triangle inequality is the key axiom for search space pruning as it circumscribes the bounds of search space during a similarity query processing. Given any three objects $o, p, q \in \mathbb{S}$, suppose that the distances $d(q, p)$ and $d(p, o)$ are known, while $d(q, o)$ is not. Then, it is not difficult to see that $d(q, o)$ can be expressed with its lower and upper bounds as $|d(q, p) - d(p, o)| \leq d(q, o) \leq d(q, p) + d(p, o)$. Any search subspaces beyond these bounds are pruned.

Metric access methods:. As described in [79], these algorithms can be roughly classified as pivot-based and compact partition algorithms.

The main idea behind *pivot-based* algorithms is to select a set of distinguished elements called *pivots* and store the distances from each pivot to the objects in the database with the goal of saving distance computations. Since different sets of pivots partition the search space in different ways, the problem of choosing a good selection of pivots is not trivial. Roughly speaking, good pivots have a high level of separation from other pivots and from other objects in the dataset. Yianilos [80] proposed the use of pivots in a tree structure called Vantage Point Tree (VPT), where an object p is chosen as a pivot and the median distance is evaluated to assign the objects with smaller (or larger) distance than the median distance to the left (or right) subspace of p, respectively. This procedure is recursively applied to obtain new pivots. Brin [81] extended this idea by using more than two pivots and assign closest points to each pivot. The recursive use of this method leads to an m-ary tree called Geometric Near-neighbor Access Tree (GNAT), being m the number of pivots considered.

On the other hand, *compact partition* algorithms attempt to obtain more local regions

than those based on pivot algorithms by means of compact regions. Ciaccia et al. [38] suggested M-tree as a metric access method with fixed-size regions, each of them represented by a central object. M-tree resembles to R-tree [37] such that all the objects are stored in the leaf nodes while the internal nodes keep pointers to the nodes at the next level. This approach has become very popular and many methods are a variant of the M-tree family. However, the overlap between regions makes M-tree works well in low and medium dimensions, but not in high dimensions. Hence, when working with high dimensional data, the overlap between regions could be so high that the idea of representing data with a hierarchy of regions is compromised when compared to a sequential search [82]. Navarro [70] reported a better result with a high dimensional data using an approach which recursively gets closer and closer to the query through neighbors in the data structure rather than using the classical divide-and-conquer pruning search of M-tree. Navarro concluded that although a graph fully exploits this technique, it presents high costs for inserting vertices and updating edges.

A.3 Hyperspherical Region Graph

As introduced in Section A.1, our strategy in finding a solution to the similarity search problem in a large metric space is to design a graph data structure. We prefer a graph-based approach mainly because additional edges in a graph allow us to traverse the data structure without backtracking to upper vertices which is required in a tree [38, 68–70]. In addition, the constraint on the root node as a starting point in a tree is relaxed in a graph because any vertex can be a starting point.

Note that the flexibility in graph traversal is enabled at the expense of additional edge insertions. Subsequently, an efficient management of edge insertion must be addressed in order to make a graph practical as an alternative to a tree data structure in similarity search. We now present the graph structure proposed in this paper, called *hyperspherical region graph* (HRG).

A.3.1 The Graph

The proposed hyperspherical region graph is a type of the relative neighborhood graph (RNG) [75] defined as follows:

Definition 5. *Given a graph $G = (V, E)$, let $H(v_i, v_j)$, for any pair of vertices $(v_i, v_j) \in V^2$, be the hypersphere of radius $d(v_i, v_j)$, for some distance function $d : V \times V \rightarrow \mathbf{R}^+$, with v_i as the center of the hypersphere. Then, G is a relative neighborhood graph if any edge $(v_i, v_j) \in E$ satisfies the* neighborhood relationship constraint*: $(H(v_i, v_j) \sqcap H(v_j, v_i)) \cap V = \emptyset$ where $H(v_i, v_j) \sqcap H(v_j, v_i)$ returns the inner volume of the lune formed by the intersection of the two hyperspheres excluding the lune's surface.*

In other words, two connected vertices are neighbors in RNG. Intuitively, RNG reduces redundant neighborhood relationships by excluding the edge between adjacent vertices v_1 and v_2 if any vertex v_3 is found in the lune formed by the intersection of the hyperspheres centered at v_1 and v_2, i.e., if the distance from v_3 to either v_1 or v_2 is smaller than the distance between v_1 and v_2.

The reason for using RNG instead of other types of graphs such as Delaunay triangulation is that RNG is defined based only on distances between vertices. This characteristic makes RNG a suitable option to perform similarity search not only in Euclidean spaces but also in arbitrary metric spaces. HRG extends the idea of RNG to represent the structure of a metric space using representative vertices of hyperspherical regions as follows:

Definition 6. *A hyperspherical region $H = (v, c, O)$ in a metric space is a hypersphere of objects $\{v, O = \{o_1, o_2, \ldots, o_n\}\}$, where (1) v is the center object of H, (2) c is the* capacity *of H, i.e., the maximum number of objects that can be included in H, and (3) O is the set of member objects that are currently contained in H (with the exclusion of v). Furthermore, the distance between v and the farthest object in H is called the* radius *of H.*

From now on, $H(v)$ will be used as a shorthand notation for $H(v, c, O)$ as long as c and O are clear in the context. Furthermore, $c_{H(v)}$ denotes c of $H(v)$, and $|H(v)| = |O|+1$.

Definition 7. *Given a set of hyperspherical regions $\{H(v_1), \ldots, H(v_n)\}$, the hyperspherical region graph $G_H = (V, E)$ is a relative neighborhood graph where $V = \{v_1, \ldots, v_n\}$, i.e.,*

the center objects of the regions, and E is the set of edges between neighboring vertices. Furthermore, $c_{H(v_1)} = \ldots = c_{H(v_n)}$

Since a hyperspherical region graph G_H is consisted of only the vertices that are centers of hyperspherical regions, not of all the data objects, the number of vertices in G_H is generally smaller than that of the RNG of the same dataset, which enables us to construct G_H by a less number of edge insertions. We now continue to discuss on the construction of a hyperspherical region graph.

In HRG construction, our goal is to minimize the update (either insertion or deletion) of the existing edges whenever we add a new object into the graph. The realization of this goal involves identifying the regions affected by vertex insertions, and hence we present the search technique used first.

A.3.2 Similarity Search in HRG: Range Search

A range search is defined by the set of objects which are at most at the distance r from the query object x.

Definition 8. *Given a universe $\mathbb{U}$ of data objects, RangeSearch* $(x, r) = \{s \in \mathbb{U} | d(x, s) \leq r\}$.

To reduce the search space in range search, the search is performed at two different levels in our approach: (1) the graph level, and (2) the region level.

Graph-level reduction. For the graph-level search space reduction, being a the starting vertex chosen randomly, the distance between x and each of the neighboring vertices of a, denoted $N(a)$, including a, is computed first. Then, a search space reduction technique, inspired by the *spatial approximation* method [70], is applied to visit only the vertices which are "close enough" to x among the vertices in $\{a\} \cup N(a)$ based on the distances. "Closeness" is defined as follows:

Definition 9. *Given a set of vertices V and the query object x, $v \in V$ is a "close-enough" vertex to x if $d(x, v) \leq min\_dist + 2r$ where $min\_dist = \min_{w \in V} d(x, w)$ and r denotes the given distance tolerance to x from v.*

This search space reduction technique is applied recursively through all the neighboring nodes that are close enough to x as long as such neighbors are found and have not been visited yet. All the visited vertices with no close-enough neighbors other than the originating neighbor are identified as candidate vertices.

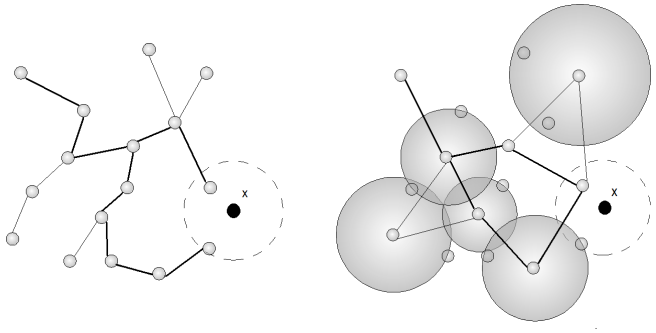
Let $\{v_1, \ldots, v_k\}$ be the set of candidate vertices of x. Then, any region $H(v_i)$ $(1 \leq i \leq k)$ which intersects that of x, i.e., $H(x)$ is returned as a candidate region because they have a higher likelihood of containing the target objects of the range search than the non-intersecting regions.

Region-level reduction. The objective of this step is to find the range search result by the given query object x and radius r. The result consists of the objects that are selected from the member objects of the candidate regions by checking if they belong to the range search region centered at x with radius r. In this process, a search space reduction is achieved by using the triangular inequality to avoid computing the distance between each member object o and x, i.e., $d(x, o)$. Note that the distance between the center object v and o is already known. The distance between x and v has been already calculated also. Hence, we can approximate our decision that o belongs to $RangeSearch(x, r)$ if the condition $|d(v, x) - d(v, o)| \leq r$ holds.

Note that it is guaranteed that for any pair of vertices v_1, v_2 in a Delaunay triangulation graph, there exists one or more paths from v_1 to v_2. Since HRG is an extension of RNG and RNG is a subgraph of Delaunay triangulation, the path of minimum length between v_1 and v_2 exists. Using both search space reduction techniques, we were able to visit 30% less edges in our approach than in SA-tree. Figure A.1 contrasts the area explored by our approach and SA-tree when performing a Range Search around the object x. The range search process is summarized in Algorithms 5 and 6.

A.3.3 Construction of HRG

HRG is a dynamic structure that allows to insert new objects at any time. Before inserting a new object x into $G_H = (V, E)$, we search a vertex $a \in V$ which is closer to x than any other vertex, starting at vertex $b \in V$. This search is known as *Nearest Neighbor*



(a) SA-tree (b) HRG

Figure A.1. Edge exploration by SA-tree and HRG. Thicker edges denote visited edges. A fewer edges are visited in HRG than in SA-tree.

Algorithm 5 RangeSearch(Object x, Range r) // x: query object, r: query radius

1: result := $\emptyset$;
2: a := random vertex in G_H;
3: $min\_dist := \infty$;
4: RangeSearch(a, x, r, min_dist, result);
5: **return** result;

Search and intuitively includes the following steps:

1. Start with a random vertex b in G_H.
2. Compute the distance between the new object x and each vertex in $N(a)$.
3. Traverse to the smallest vertex $a_i \in N(a)$, and repeat from Step 2 as long as a vertex with a shorter distance is found.

As soon as such nearest vertex a is found, x is inserted into $H(a)$ rather than being inserted as a new vertex in G_H. In such a case, we check the cardinality, $|H(a)|$, of the updated $H(a)$. If $|H(a)|$ is less than or equal to the capacity of $H(a)$, c, the insertion operation of x is completed. If $H(a)$ is already saturated to its capacity, $H(a)$ is divided into two regions $H'(a)$ and $H(y)$ such that one of the farthest objects y of $H(a)$ becomes a new vertex in G_H, and all other objects in $H(a)$ are retained in $H'(a)$. Subsequently, a

Algorithm 6 RangeSearch (Vertex a, Object x, Range r, Distance min_dist, Stack S)

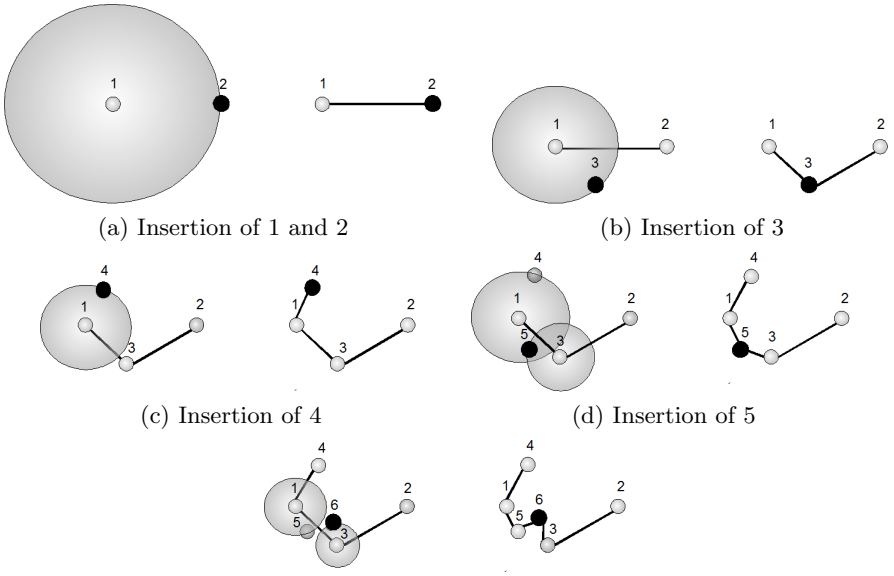
```
1: if a has not been visited yet then
2:   if d(a,x) − r ≤ radiusRegion(a) ∨ d(a,x) + r ≤ radiusRegion(a) then
3:     for each object o in H(a) do
4:       if |d(a,x) − d(o,a)| ≤ r then
5:         if calculateDistance(o,x) ≤ r then
6:           result := result ∪ {o};
7:         end if
8:       end if
9:     end for
10:  end if
11:  min_dist := min{{min_dist} ∪ {d(b,x) : b ∈ N(a)}};
12:  for b ∈ N(a) do
13:    if d(b,x) ≤ min_dist + 2r then
14:      mark b as VISITED;
15:      RangeSearch(b, x, r, min_dist, result);
16:    end if
17:  end for
18: end if
```

new region $H(y) = (y, c_{H(a)}, \{\})$ is generated. Note that the capacity of $H(y)$ is set to the same capacity of $H(a)$. This process is summarized recursively in Algorithm 7.

Note that the addition of a new vertex in G_H, subsequently the addition of the corresponding new region, requires the update of the neighborhood relationships among the vertices because the relationships may change according to the neighborhood constraint of RNG. This can be as simple as inserting an edge between the new vertex x and the existing vertex from which x was born. However, x may change the existing neighborhood relationships.

We now compare the constructions of a hyperspherical region graph G_H of capacity 2 and a relative neighborhood graph G_R step by step in the following example using Figure A.2. In the example, $\eta(a, b)$ denotes the neighborhood relationship between vertices a and b.

Example 3. *Each object (either vertex or non-vertex) in Figure A.2 is labeled according to their insertion order. The lefthand side of each subfigure shows G_H whereas the righthand side G_R. Figure A.2(a) shows the first two objects 1 and 2 are inserted into G_H and G_R*



(a) Insertion of 1 and 2 (b) Insertion of 3

(c) Insertion of 4 (d) Insertion of 5

(e) Insertion of 6

Figure A.2. A step-by-step construction comparison of our HRG and the RNG. The left graph of each subfigure is the HRG while the right is the RNG. During the insertion of the six objects, HRG modified five edges whereas RNG 11 edges. This saving is proportional to the capacity of the hyperspheres.

respectively. Because the capacity of G_H *is 2, object 2 becomes a member object of the region centered at object 1,* $H(1)$*. Object 1 becomes the only vertex in* G_H *at the moment. In contrast, two objects 1 and 2 become neighboring vertices in* G_R *and hence* $\eta(1,2)$ *holds.*

As object 3 is inserted into $H(1)$ *due to its closer distance to object 1 than object 2, object 2 is separated from* $H(1)$ *to form a new vertex in* G_H*. Hence,* $\eta(1,2)$ *is established. On the other hand in* G_R*,* $\eta(1,2)$ *is revoked, and* $\eta(1,3)$ *and* $\eta(2,3)$ *are added because the neighborhood constraint defined in Def. 5.*

In Figure A.2(c), since $d(1,4) < d(1,3)$*, object 3 becomes a new vertex in* G_H *and object 4 becomes a member of* $H(1)$*. In Figure A.2(d), object 5 is inserted into* $H(3)$ *due to its distance to vertex 3.*

Now consider the evolution of G_H *in Figure A.2(e) in which object 6 is inserted into* $H(3)$*. Due to this update, object 5 is expelled from* $H(3)$ *and inserted into* $H(1)$*, which has a cascading effect on object 4. In other words, object 4 is expelled from* $H(1)$ *because*

$d(1,4) > d(1,5)$.

HRG minimizes the insertion of edges between vertices by storing non-vertex objects in regions without creating edges. For example, after the six objects are inserted in Figure A.2, HRG with capacity of 2 modifies only 5 edges (4 insertions and 1 deletion) whereas RNG modifies 11 edges (8 insertions and 3 deletions), leading to 55% less edge update in HRG.

Insertion of a new vertex into G_H may have an impact on the neighborhood relationships in G_H since G_H is a type of RNG. Clearly, we want to avoid to reevaluate the entire neighborhood relationships each time we add a vertex, but update only the neighborhood relationships affected by the new vertex. This strategy is called *LocalInsert* in our algorithm.

LocalInsert. Given the new vertex x which is the center of the region having no member objects yet, *LocalInsert*() first identifies the vertices which are located within certain distance r from x. The distance r is determined based on the distance from the new vertex x to the center of the closest region $H(a)$, i.e., $d(a,x)$, and another distance to the farthest object f in $H(a)$, i.e., $d(f,x)$. We want to scale this distance by a tolerance parameter ϵ. If ϵ is set to a larger value, we can obtain a more global solution with the price of high edge update cost. On the other hand, with a smaller ϵ, we can achieve more cost effective edge update with potentially less optimal graph structure. Practically, a real number in the range $[0-1]$ is chosen for ϵ.

By using the range query with the distance r as the radius, we obtain existing vertices in the graph which are potentially neighbors to the new vertex x. For this set of vertices, actual neighborhood relationships are updated. By employing this strategy, we are able to achieve a better performance than Incremental-RNG [71]. The process of inserting a new vertex in G_H is summarized in Algorithm 8.

A.4 Experimental Results

We tested the proposed HRG in terms of construction and similarity search operations using two real and two synthetic datasets.

Algorithm 7 Insert (Vertex b, Object x) // b: starting location, x: new object

```
1: a := the vertex of the region where the nearest neighbor to x is placed;
2: if |H(a)| < c then
3:    H(a) := H(a) ∪ {x}; // x becomes a member object of H(a)
4:    save distance d(a, x);
5: else if d(a, x) > regionRadius(H(a)) then
6:    LocalInsert(a, x); // insert x as a new vertex
7: else
8:    H(a) := H(a) ∪ {x}; // x becomes a member object of H(a)
9:    x := farthestObject(H(a)); // x: farthest object in H(a)
10:   H(a) := H(a) − {x};
11:   LocalInsert(a, x);
12: end if
```

Algorithm 8 LocalInsert (Vertex a, Vertex x)

```
1: f := farthestObject(H(a));
2: r := (d(a, x) + d(f, x)) * (1 + ε);
3: result := RangeSearch(x, r); // result: set of vertices
4: buildRNG(result ∪ {x});    // x becomes a new vertex; update neighborhood
```

1. HSV-3: This is a real dataset consisted of 15,000 3-dimensional objects obtained from a color image by sampling the hue (H), saturation (S), and value (V) of every three other pixel.

2. Faces-175: We generated this real dataset consisted of 600 175-dimensional objects from 600 face images found in the CMU Face Database.[1] Each face figure is divided into $5 \times 5 = 25$ regions and the following seven features are extracted from each region: the average and the standard deviation for each of H, S and V, and the percentage of the presence of skin color in a region, which yield 600 feature vectors of $7 \times 5 \times 5 = 175$ dimensions.

3. Synthetic-2: This synthetic dataset contains 1,000 2-dimensional vectors normally distributed in 10 clusters with overall standard deviation of 0.1 over a rectangle with a minimum and maximum value of 0 and 1 on each dimension respectively.

[1] `http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-8/ml94faces/faces`

4. Synthetic-16: This synthetic dataset contains 1,500 16-dimensional vectors normally distributed in 10 clusters with overall standard deviation of 0.1 over a hypercube with a minimum and maximum value of 0 and 1 on each dimension respectively.

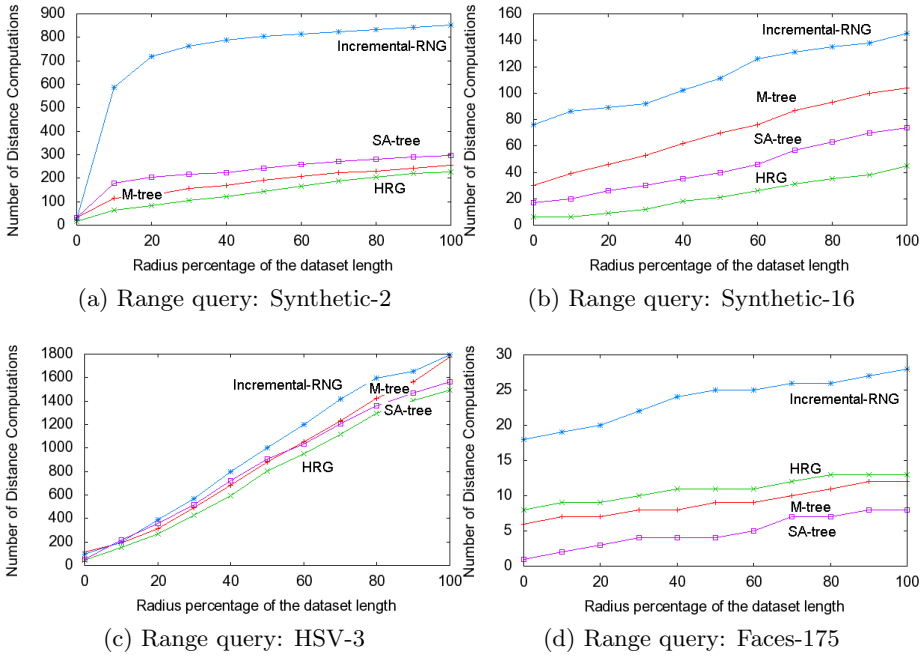
The performance of HRG was then compared to those of the two most well known tree structures M-tree [38] and SA-tree [70], and a graph data structure Incremental-RNG [71] in terms of the number of distance computations, total response time, and memory usage.

1. Number of distance computations (NDC): Whenever a new object is to be inserted or a similarity query is performed, a number of distance computations are to be performed between the new/query object and some of the existing vertices in the graph. In fact, the number of times two objects are compared in a data structure is a measure independent of the hardware or software platform. For this reason, NDC has been a traditional measure to evaluate the performance of data structures and similarity search methods in metric spaces [38, 68, 69, 77, 79].

2. Response time: While NDC is one major factor contributing to the time complexity of an algorithm, the response time is useful to evaluate the overall system performance of each approach in comparison.

3. Memory usage: An algorithm may be able to achieve a high performance while requiring a significant amount of main memory. We believe that space requirement should also be taken into consideration for a fair evaluation.

The Euclidean distance (L_2) was used as the distance function in our experiments. Furthermore, for similarity search, we used the range query to make our results comparable to others since the experimental results of other approaches found in the literature are based on this type of search. All the algorithms were implemented in Java 1.6 and tested on a 2.0 GHz PC with 2038 MB of RAM running 32-bit Microsoft Windows Vista.



(a) Range query: Synthetic-2 (b) Range query: Synthetic-16

(c) Range query: HSV-3 (d) Range query: Faces-175

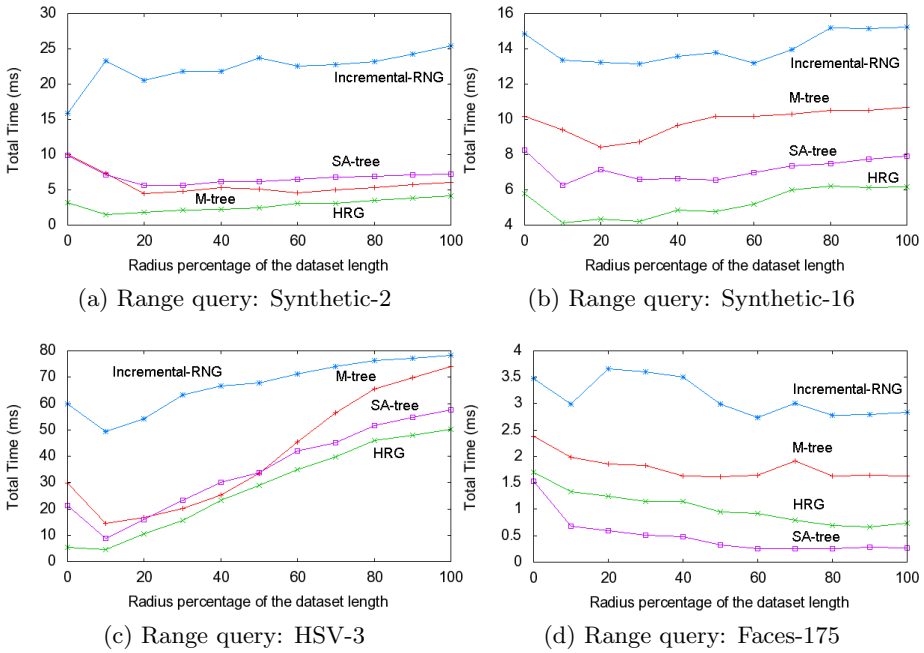
Figure A.3. Comparison of HRG by number of distance computations in range search. Note that the scale of y-axis is different from one figure to another.

A.4.1 Similarity Search Performance

NDC. In this test, the data structures in comparison were tested with different range search radii, ranging from 10% to 100% of the maximum distance[2] between the two farthest objects in the dataset. HRG performed better than all other structures with Synthetic-2, Synthetic-16 and HSV-3, as shown in Figures A.3(a), A.3(b) and A.3(c). With Faces-175, HRG outperformed Incremental-RNG with a comparable performance with M-tree (see Figure A.3(d)). In case of Faces-175, potentially high overlap among the regions in HGR offsets the anticipated benefit of HRG. In case of such high dimensional data, we observed that SA-tree is the best out of the structures in comparison.

Response time. This test was conducted in a similar way to the above to measure the total evaluation time of range searches. All the test dataset was fit into the main memory. The performance of HRG in this test was congruent with the result of NDC,

[2]This distance is referred as the dataset length in Figures A.3, A.4 and A.5.



(a) Range query: Synthetic-2 (b) Range query: Synthetic-16

(c) Range query: HSV-3 (d) Range query: Faces-175

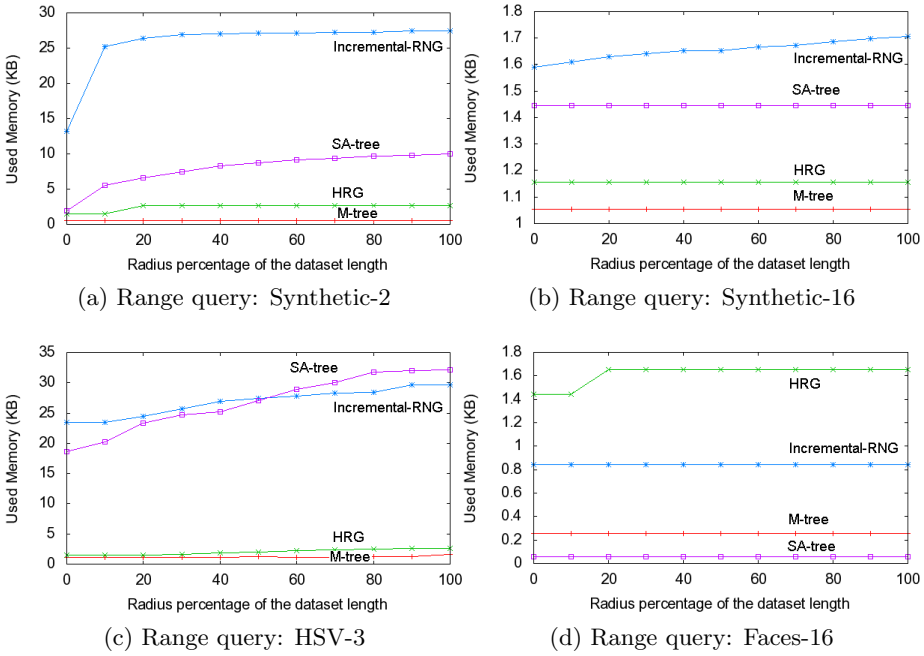
Figure A.4. Comparison of HRG by total range query evaluation time at various radii.

which was anticipated because the main cost during running time is the computation of distances between objects, as the result is shown in Figure A.4.

Memory usage. In this test for the total memory requirement during range searches, in contrast to the previous tests, M-tree and HRG used less memory than the others in case of Synthetic-2 and HSV-3, as shown in Figures A.5(a) and A.5(c), since both data structures store in memory the precomputed distances between vertices and the objects inside the associated regions. In case of Figures A.5(b) and A.5(d), note that the scale of y-axis is relatively small.

A.4.2 Construction Performance

In this test, HRG consistently performed better than Incremental-RNG in all the cases. In comparison with the construction time of the tree indices, however, HRG generally was not able to perform better than the trees due to neighborhood update on extra edges. (Due to the page limit, the detail figures are omitted.)



(a) Range query: Synthetic-2

(b) Range query: Synthetic-16

(c) Range query: HSV-3

(d) Range query: Faces-16

Figure A.5. Comparison of HRG by memory usage during similarity query. Note that the scale of y-axes of (b) and (d) is substantially smaller than that of (a) and (c).

A.5 Conclusions and Future Work

This paper introduced a graph-based indexing algorithm and discussed its performance for the similarity search problem in metric spaces. The proposed algorithm, which uses a small representation of the search space, can be classified as a compact partitioning algorithm. The notion of neighborhood present in the spatial approximation search technique [70] is fully exploited in the graph. Moreover, since each vertex is the center of a hyperspherical region, our approach only visits the neighbors which are close enough to the query object, thus saving the evaluation of distances between the query object and other objects.

The experimental results showed that the proposed method performs better than other tree and graph-based approaches in similarity search with datasets of moderate dimensionality. In case of a high dimensional dataset, we observed overlap between regions, which offsets the benefits of the proposed structure.

Clearly, the region overlaps observed in a high dimensional dataset needs to be addressed in the future work. More complex objects also need to be considered in order to further validate the benefits of using the proposed graph structure.

CURRICULUM VITAE

Omar U. Florez

EDUCATION

Ph.D., Computer Science. Utah State University, Logan, UT. 2013.

B.S., Computer Science. San Agustin University, Peru 2007.

RESEARCH INTERESTS

Large scale machine learning, databases, cross-domain content recommendation, data mining, data analytics and computer vision

HONORS

IBM Innovation Award on Scalable Data Analytics. 2010.

Best Research Poster in the 2013 ACM Tapia Celebration of Diversity in Computing. 2013

Google Scholarship for the Society of Hispanic Professional Engineers (SHPE) national conference. 2012.

Chevron and SHPE Academic Scholarship for the Society of Hispanic Professional Engineers (SHPE) - Region 3. 2012.

Best Research Poster in the NSF Workshop for Underrepresented Senior Graduate Students at Georgia Tech. 2012.

Semi-finalist in the global startup competition Intel Challenge with PrimerosPuestos.org. 2011.

CONFERENCE PUBLICATIONS

Omar U. Florez and Curtis Dyreson, *What to reuse?: A probabilistic model to transfer user annotations in a surveillance video*, Submitted to the 19th Conference on Knowledge Discovery and Data Mining (2013 ACM SIGKDD).

Curtis Dyreson, **Omar U. Florez**, and Akshay Thakre, *Supporting Data Aspects in Pig Latin*, in Proceedings of the Aspect-Oriented Software Development (AOSD 2013), Fukuoka, Japan.

Omar U. Florez and Curtis Dyreson, *Is that scene dangerous?: transferring knowledge over a video stream*, in Proceedings of the Ph.D. workshop in the CIKM conference (CIKM 2012), Hawaii, USA.

Omar U. Florez and Curtis Dyreson, *Discovering Activity Interactions in a Single Pass over a Video Stream*, in Proceedings of the ACM Symposium on Applied Computing (ACM SAC 2012), Trento, Italy.

Curtis Dyreson and **Omar U. Florez**, *Building a Display of Missing Information in a Data Sieve*, in Proceedings of the ACM 14th International Workshop On Data Warehousing and OLAP (DOLAP) colocated with ACM CIKM 2011, Glasgow, Scotland, UK.

Omar U. Florez and Curtis Dyreson, *Scalable Similarity Search of Timeseries with Variable Dimensionality*, in Proceedings of the 20th ACM Conference on Information and Knowledge Management (ACM CIKM 2011), Glasgow, Scotland, UK.

Curtis Dyreson and **Omar U. Florez**, *Data Aspects in a Relational Database*, in Proceedings of the 19th ACM Conference on Information and Knowledge Management (ACM CIKM 2010), Toronto, Canada.

Omar U. Florez and Curtis Dyreson, *Mining Rules to Explain Activities in Videos*, in Proceedings of the 19th ACM Conference on Information and Knowledge Management (ACM CIKM 2010), Toronto, Canada.

Omar U. Florez, Alexander Ocsa, and Curtis Dyreson, *Sublinear Similarity Search of Realistic Timeseries and its Application to Human Motion*, in Proceedings of the 11th ACM International Conference on Multimedia Information Retrieval (MIR 2010), Pennsylvania, USA.

Omar U. Florez, Xiaojun Qi, and Alexander Ocsa, *MOBHRG: Fast K-Nearest Neighbor Search by Overlap Reduction of Hyperspherical Regions*, in Proceedings of the 34th International ACM Conference on Acoustics, Speech, and Signal Processing (ICASSP 2009), Taipei, Taiwan.

Omar U. Florez and SeungJin Lim, *Discovery of Interpretable Time Series in Video Data Through Distribution of Spatiotemporal Gradients*, in Proceedings of the 24th Annual ACM Symposium on Applied Computing (ACM SAC 2009), Hawaii, USA.

Omar U. Florez and SeungJin Lim, *HRG: A Graph Structure for Fast Similarity Search in Metric Spaces*, in Proceedings of the 19th International Conference on Database and Expert Systems Applications (DEXA 2008), Turin, Italy.

Omar U. Florez and Ernesto Cuadros-Vargas, *Improving Human Computer Interaction through Spoken Natural Language*, in Proceedings of the IEEE Symposium on Computational Intelligence in Image and Signal Processing, Hawai, USA. 2007.

Omar U. Florez and Michael G. Hinchey, *Asteroid Exploration with Autonomic Systems, a Biologically-inspired approach based on the Immunological System*, in Proceedings of the 4th IEEE Workshop on Engineering of Autonomic and Autonomous Systems, Baltimore, USA. 2007.

Omar U. Florez, *Voice2SQL: Hybrid Intelligent System for the recovery of information from databases by means of Spoken Natural Language*, in Proceedings of the 2nd International IEEE Conference on Intelligent Information Hiding and Multimedia Signal Processing, California, USA. 2006.

Omar U. Florez and Ernesto Cuadros-Vargas, *A Biologically Motivated Computational Architecture Inspired in the Human Immunological System to Quantify Abnormal Behaviours to Detect Presence of Intruders*, in Proceedings of the 19th IFIP World Computer Congress, IFIP International Conference on Biologically Inspired Cooperative Computing (WCC 2006), Santiago, Chile.

Omar U. Florez, Jose Mercado Ticona and Yordan Yampi Enciso, *Recovery by similarity of shape of fingerprints by means of an indexation in metric space and image enhancement by using Fourier transform*, in Proceedings of the 32nd Latin-American Conference on Computer science (CLEI 2006), Santiago, Chile. 2006.

Omar U. Florez, *Quantifying the presence of intruders through software metrics by means of a model based on the Human Immunologic System*, in Proceedings of the 5th Ibero-American Conference of Software Engineering and Knowledge Engineering. ISBN: 970-94770-0-5, Puebla, Mexico. 2005.

JOURNALS

Omar U. Florez and Curtis Dyreson, *One-Scan Rule Extraction to Explain Significant Vehicle Interactions with Guaranteed Error Value*, ACM SIGAPP Applied Computing Review. 2012.

Omar U. Florez and SeungJin Lim, *A Spoken Natural Language-Based Interface for Querying SQL Databases*, in International Journal of Information Technology and Intelligent Computing, Int. J. ITIC. 2007.

BOOK CHAPTERS

Omar U. Florez and SeungJin Lim, *Modeling Query Events in Spoken Natural Language for Human-Database Interaction*, in Human Computer Interaction: New Developments. Chapter 13, ISBN: 978-3-902613-38-7, InTech Education and Publishing. Vienna, Austria. 2008.

Omar U. Florez and SeungJin Lim, *Evaluation of Clustering on Self-Organizing Maps, To appear in Self-Organizing Maps*, InTech Education and Publishing. ISBN: 978-953-7619-X-X, InTech Education and Publishing. 2009.

POSTERS

Omar U. Florez and Curtis Dyreson, *Is that scene dangerous?: transferring knowledge over a video stream*, in ACM Tapia Celebration of Diversity in Computing Conference, (Best Poster). 2012.

Omar U. Florez and Curtis Dyreson, *Automatic Explaining Vehicle Interaction in Congested Roads*, in NSF Academic Workshop in Computing for Underrepresented Ethnic Minorities at the level of Assistant Professor, Associate Professor, and Senior Doctoral Student, (Best Poster). 2011.

INVITED TALKS

Multimedia Personalization with Common Latent Variables, Pandora Inc. 2013.

A Cross-domain Recommender System, Intel Labs. 2012.

Massive Rule Extraction to Explain Significant Vehicle Interactions with Guaranteed Error Value, General Electric Global Research Center. 2012.

Content Protection for Medical Videos, IBM Almaden Research Center. 2011.

Advanced Similarity Search of Realistic Timeseries, Computer Science department, University of California, Riverside. 2011.

INDUSTRY EXPERIENCE

Software Engineer, Intern. Intel Labs. Summer 2012.

Research Scientist, Intern. IBM Research. Summer 2011.

Research Scientist, Intern. IBM Research. Summer 2010.

SERVICE

Co-Founder of www.PrimerosPuestos.org. December 2010.

Founder of South Americans in Computing. August 2011.

Member of the Society of Hispanic Professionals. August 2011.

Member of the Peruvian Computer Society. January 2007.